SOLOQ: CALIBRATION-FREE QUANTIZATION FOR DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion large language models (dLLMs) have emerged as a promising alternative to autoregressive language models through bidirectional diffusion-based token generation. However, their growing model sizes and high inference costs make efficient deployment challenging: full-sequence denoising repeatedly invokes compute-intensive forward passes, while block-diffusion models additionally introduce a memory-intensive KV-cache. Low-bit weight–activation quantization is therefore attractive, yet existing dLLM post-training quantization methods rely on calibration data despite activation distributions shifting across masking states and denoising steps. We present **SOLOQ**, a calibration-free quantization framework that maps weights and activations into a normalized rotated basis with a predictable marginal distribution, enabling data-independent quantization. SOLOQ combines a structured K-RPBH rotation with a lightweight rescaling correction for calibration-free quantization. Its predictable post-rotation distribution supports both distribution-matched codebooks and hardware-native NVFP4. For block-diffusion models, SOLOQ further applies commit-time KV-cache quantization to compress persistent states without perturbing the actively denoised block. Across full-sequence dLLMs (LLaDA and Dream) and block-diffusion dLLMs (Fast-dLLM v2 and Nemotron-Labs-Diffusion), SOLOQ retains accuracy under 4-bit quantization and outperforms calibration-based baselines on knowledge- and reasoning-intensive benchmarks. With NVFP4, SOLOQ reduces peak memory by up to $2.61\times$ and accelerates end-to-end inference by up to $2.24\times$.

1 INTRODUCTION

Large language models (LLMs) have achieved remarkable progress across a wide range of language understanding, reasoning, and generation tasks (Brown et al., 2020; Achiam et al., 2023; Touvron et al., 2023; Team et al., 2023), with autoregressive (AR) modeling remaining the dominant paradigm for text generation. In parallel, diffusion models (Bao et al., 2023; Peebles & Xie, 2023) have become a powerful generative framework for continuous domains such as image and video generation (Esser et al., 2024; Wu et al., 2025; Kong et al., 2024; Yang et al., 2025), where iterative denoising enables high-quality and flexible generation. Motivated by these advances, recent diffusion large language models (dLLMs) extend diffusion-based generation to discrete text, replacing strictly left-to-right decoding with iterative masked denoising and bidirectional context. Representative models such as LLaDA (Nie et al., 2026) and Dream (Ye et al., 2025) have demonstrated competitive language understanding, reasoning, and generation capabilities, suggesting that strong language modeling is not inherently tied to AR decoding. More recently, block-diffusion models (Wu et al., 2026a; Fu et al., 2026; Team et al., 2026) further bridge diffusion-based generation and practical inference by generating multiple tokens within a block and reusing previously generated context through caching. Together, these developments establish diffusion-based language modeling as a promising alternative to conventional AR generation.

Despite this progress, efficient dLLM inference remains challenging as model size and context length grow. Unlike AR decoding, dLLMs iteratively refine multiple tokens in parallel across denoising steps, while long-context inference increases the memory and computation associated with activations and KV-cache. To better understand these bottlenecks, we profile Fast-dLLM v2 (Wu et al., 2026a) from memory and latency perspectives, as shown in Figure 1(a). At short contexts, weights dominate the memory footprint, but the relative contribution of activations and the KV-cache grows with context length. In terms of latency, attention becomes expensive as the sequence length grows. In dLLMs, these costs are repeatedly incurred across denoising steps, making weight-only

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

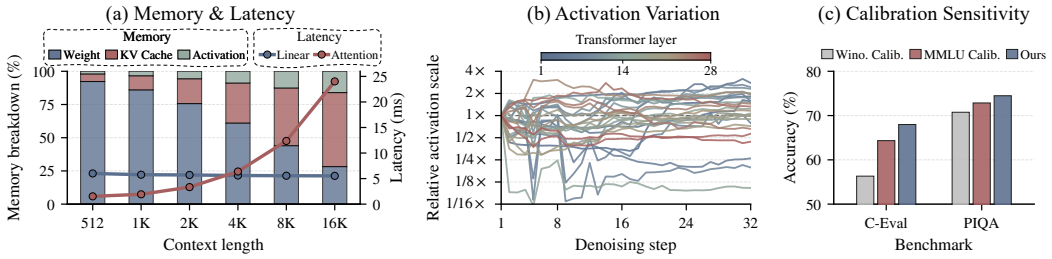


Figure 1: Profiling and motivation for SoloQ. (a) Memory breakdown at batch size 128 and aggregate linear/attention latency at batch size 1 for Fast-dLLM v2 (Wu et al., 2026a) across context lengths on an H200 GPU. (b) Layer-wise variation in `down_proj` input activation scale across denoising steps, normalized to the first step. (c) Accuracy of PTQ (Ashkboos et al., 2024; Frantar et al., 2022) with WinoGrande/MMLU calibration versus calibration-free SoloQ.

compression insufficient for efficient inference. Instead, low-bit quantization of weights, activations, and KV-cache can reduce both computation and memory across context regimes.

Post-training quantization (PTQ) has been widely studied for AR LLMs to reduce memory and computation without retraining. Existing LLM PTQ methods span several complementary paradigms, including second-order methods that exploit Hessian information for low-bit weight quantization (Frantar et al., 2022; Li et al., 2025; Lee et al., 2026b), scaling- and salience-based methods that mitigate activation outliers through scaling or reparameterization (Xiao et al., 2023; Wei et al., 2023; Shao et al., 2024), and rotation-based methods that transform weights and activations into more quantization-friendly representations (Chee et al., 2023; Ashkboos et al., 2024; Liu et al., 2025; Hu et al., 2025). These methods enable efficient low-bit inference while preserving accuracy.

However, quantization on dLLMs is more challenging than AR LLMs. During diffusion decoding, the model repeatedly encounters different masking states and denoising steps, leading to substantial variation in activation distributions and quantization sensitivity throughout generation, as shown in Figure 1(b). Existing dLLM PTQ methods (Zhang et al., 2026; Xu & Yang, 2025; Yan et al., 2026; Huang et al., 2026) rely on calibration data to capture or compensate for these state- and time-dependent variations. Figure 1(c) illustrates this sensitivity for a representative calibration-based PTQ method (Ashkboos et al., 2024; Frantar et al., 2022): changing calibration data from WinoGrande to MMLU alters accuracy across the benchmarks, showing that calibration choices may not generalize across tasks. This motivates a different direction: *rather than tracking a changing distribution, can we transform it into a predictable form that can be quantized without calibration?*

To realize this principle, we propose **SoloQ**, a calibration-free dLLM quantization framework that avoids calibration-set dependence while maintaining strong performance, as shown in Figure 1(c). Rather than adapting to activation distributions that vary across masking states and denoising steps, SoloQ transforms weights and activations into a normalized rotated space with a predictable dimension-dependent marginal distribution. This enables data-independent quantization that can be shared across diffusion states without calibration. To support arbitrary transformer dimensions, SoloQ introduces a Kronecker-extended randomized permuted block-Hadamard (K-RPBH) rotation that combines efficient local mixing with global cross-block mixing. We further introduce a scale-only bias correction that restores the radial component of low-bit quantized representations with negligible overhead. For block-diffusion models, SoloQ extends the same calibration-free design to persistent KV states through commit-time KV-cache quantization. Built on these components, SoloQ supports both distribution-matched codebook quantization and hardware-native NVFP4 without chasing activation distributions. Our contributions are summarized as follows:

- We introduce **SoloQ**, a calibration-free dLLM quantization framework combining K-RPBH rotation with a lightweight scale-only bias correction.
- The resulting single predictable distribution supports both distribution-matched codebooks and hardware-native NVFP4 within the same calibration-free framework.
- We extend SoloQ to block-diffusion models through commit-time KV-cache quantization, covering weights, activations, and persistent KV states.
- Across full-sequence and block-diffusion dLLMs, SoloQ achieves strong 4-bit accuracy with up to a $2.61\times$ reduction in peak memory and $2.24\times$ faster end-to-end inference.

2 Related Work

2.1 Diffusion Large Language Models

Diffusion-based text generation has evolved from early discrete and masked diffusion models (Austin et al., 2021a; Li et al., 2022; He et al., 2023) to large-scale diffusion language models that perform iterative denoising with bidirectional context. Recent full-sequence models such as LLaDA (Nie et al., 2026), LLaDA-1.5 (Zhu et al., 2026), and Dream (Ye et al., 2025) demonstrate competitive language understanding, reasoning, and generation at billion-parameter scale. Beyond full-sequence diffusion, block-based and hybrid approaches improve decoding efficiency by generating multiple tokens per block and reusing previously generated context, including Block Diffusion (Arriola et al., 2025), Fast-dLLM (Wu et al., 2026b), Fast-dLLM v2 (Wu et al., 2026a), and LLaDA2.X (Bie et al., 2025; 2026). More recent systems further broaden this design space: Nemotron-Labs-Diffusion (Fu et al., 2026) unifies autoregressive, diffusion, and self-speculative decoding, Nemotron-TwoTower (Reda et al., 2026) separates autoregressive context modeling from diffusion denoising, and DiffusionGemma (Team et al., 2026) scales block-diffusion generation to a 26B mixture-of-experts model. These developments highlight the rapid progress of diffusion-based language modeling toward scalable and practical text generation.

2.2 Calibration-Free and Data-Independent Quantization

A complementary line of research avoids calibration altogether by deriving quantizers from analytically known or data-independent distributions. TurboQuant (Zandieh et al., 2026) randomly rotates input vectors to induce a known coordinate distribution, enabling coordinate-wise optimal quantization. PolarQuant (Han et al., 2025) similarly exploits an analytically known distribution after random preconditioning and polar transformation for online vector and KV-cache quantization. OrbitQuant (Lee et al., 2026a) extends this principle to image and video diffusion transformers using randomized permuted block-Hadamard rotations for calibration-free weight–activation quantization. ConvRot (Huang et al., 2025) applies group-wise Hadamard rotations for W4A4 diffusion-transformer quantization, while DVD-Quant (Li et al., 2026) combines data-free grid refinement, rotation, and adaptive bit-width allocation for video diffusion transformers. SoloQ builds on this distributional perspective for diffusion language models, extending calibration-free quantization to arbitrary transformer dimensions and persistent KV states in block-diffusion models.

2.3 Post-Training Quantization for dLLMs

Recent work has begun to develop PTQ methods specifically for diffusion language models. Quant-dLLM (Zhang et al., 2026) introduces masked calibration simulation for timestep-dependent masking and targets extreme low-bit weight quantization. DLLMQuant (Xu & Yang, 2025) accounts for temporal and masking variations for weight–activation quantization, while STaR-Quant (Yan et al., 2026) addresses state-dependent activation disparity and temporal error accumulation. FAIR-Calib (Huang et al., 2026) improves calibration by emphasizing fragile token-commitment states. Despite their designs, existing dLLM PTQ methods remain calibration-based and focus on full-sequence diffusion models. To our knowledge, SoloQ is the first calibration-free framework to support KV-cache quantization for block-diffusion models while also covering full-sequence dLLMs.

3 Preliminary

3.1 Diffusion Language Models

Let $x_0 = (x_0^1, \ldots, x_0^L) \in \mathcal{V}^L$ denote a clean token sequence of length L. At diffusion timestep $t \in [0, 1]$, each token is independently preserved with probability α_t or replaced by a special mask token M:

$$q_{t|0}(x_t \mid x_0) = \prod_{i=1}^{L} \left[\alpha_t \, \delta_{x_0^i}(x_t^i) + (1 - \alpha_t) \, \delta_{\mathrm{M}}(x_t^i) \right], \tag{1}$$

where x_t is the partially masked sequence and $\delta_a(\cdot)$ denotes a point mass at token a. The denoising model predicts masked tokens from bidirectional context,

$$p_\theta(x_0^i \mid x_t), \qquad i \text{ s.t. } x_t^i = \mathrm{M}, \tag{2}$$

and progressively unmasks the sequence over multiple steps. Full-sequence dLLMs (Nie et al., 2026; Ye et al., 2025) repeatedly process the entire sequence, whereas block-diffusion models (Wu et al., 2026a; Fu et al., 2026) denoise one block at a time while reusing KV states from previously committed blocks.

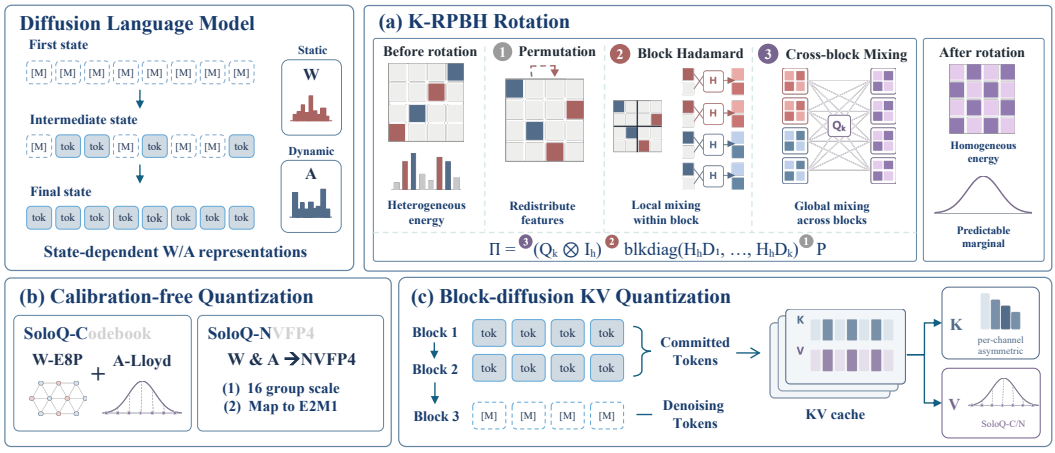


Figure 2: **Overview of SoloQ.** (a) K-RPBH combines permutation, block-Hadamard, and cross-block mixing to transform heterogeneous representations into a predictable space. (b) SoloQ-C and SoloQ-N enable calibration-free weight–activation quantization using distribution-matched codebooks and NVFP4, respectively. (c) For block-diffusion models, SoloQ applies commit-time KV quantization, keeping the denoising block in full precision.

3.2 Predictable Marginals under Random Rotation

Consider a nonzero vector $x \in \mathbb{R}^d$ with normalized direction $\tilde{x} = x/\|x\|_2$. Under a Haar-random rotation $\mathbf{R}$, $z = \mathbf{R}\tilde{x}$ is uniformly distributed on the unit sphere, independent of the direction of x (Zandieh et al., 2026). Consequently, each coordinate follows the fixed spherical marginal

$$f_d(u) = \frac{\Gamma(d/2)}{\sqrt{\pi}\,\Gamma((d-1)/2)}(1-u^2)^{(d-3)/2}, \quad u \in [-1,1] \approx \mathcal{N}\left(0, \frac{1}{d}\right), \tag{3}$$

where the Gaussian approximation becomes increasingly accurate as d grows. Thus, random rotation maps arbitrary normalized representations to a predictable Gaussian-like distribution, enabling analytic quantizer design without calibration. OrbitQuant (Lee et al., 2026a) realizes this principle using randomized permuted block-Hadamard (RPBH) rotations. For $d = kh$, with Hadamard block size h and k blocks, RPBH applies a global random permutation followed by independent randomized Hadamard transforms within each block, providing $O(d \log h)$ mixing without dense-rotation cost. However, its blockwise structure can leave residual inter-block energy imbalance, which becomes more pronounced as the number of blocks grows due to limited cross-block mixing.

4 Method

4.1 Overview

Rather than tracking activation distributions that shift across masking states and denoising steps, SoloQ transforms representations into a predictable rotated space for low-bit quantization. It applies K-RPBH for global mixing (Section 4.2), followed by distribution-matched codebooks (SoloQ-C) or hardware-native NVFP4 (SoloQ-N) (Section 4.3) and a lightweight scale-only correction (Section 4.4). For block-diffusion models, SoloQ further extends this design to persistent KV states through commit-time quantization (Section 4.5). Figure 2 illustrates the overall pipeline.

4.2 K-RPBH Rotation

Rotation plays a key role in calibration-free quantization by mapping normalized representations to a predictable, approximately Gaussian coordinate distribution, as shown in Eq. 3. RPBH, introduced in OrbitQuant (Lee et al., 2026a), achieves this efficiently, but has two limitations for heavy-tailed dLLM representations. **(1) Outlier-sensitive block balancing:** RPBH relies on the initial permutation to balance the energy assigned to each block, and its variance-concentration guarantee weakens as the largest input coordinate grows. **(2) No cross-block correction:** Once the permutation is fixed, each Hadamard transform operates independently within its block, so any residual block-energy imbalance cannot be redistributed across blocks. This effect is particularly relevant for dimensions composed of many Hadamard blocks, such as Dream's feed-forward network (FFN) dimension $d = 18944 = 512 \times 37$.

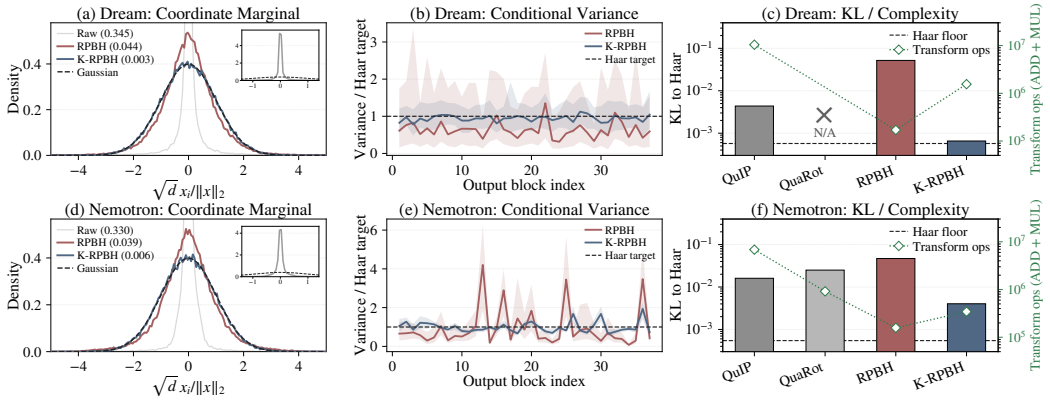


Figure 3: **Effect of K-RPBH rotation on dLLM activations.** Top row shows Dream-7B layer-1 FFN down projection and bottom row shows Nemotron layer-2 FFN down projection. (a,d) Coordinate marginals before and after rotation. (b,e) Conditional coordinate variance across output blocks, normalized by the Haar target $1/d$. (c,f) KL divergence to the Haar reference versus transform complexity. N/A denotes an unsupported transform dimension for QuaRot (Ashkboos et al., 2024).

To address these limitations, we introduce the Kronecker-extended randomized permuted block-Hadamard (K-RPBH) rotation,

$$\mathbf{\Pi} = (\mathbf{Q}_k \otimes \mathbf{I}_h)\,\mathrm{blkdiag}\,(\mathbf{H}_h\mathbf{D}_1, \ldots, \mathbf{H}_h\mathbf{D}_k)\,\mathbf{P}, \tag{4}$$

where $d = kh$ with h a power of two, $\mathbf{H}_h \in \mathbb{R}^{h\times h}$ is the normalized Walsh–Hadamard matrix, $\mathbf{D}_i \in \mathbb{R}^{h\times h}$ is a diagonal matrix with i.i.d. Rademacher entries, $\mathbf{P} \in \mathbb{R}^{d\times d}$ is a uniformly random permutation matrix, $\mathbf{Q}_k \in \mathbb{R}^{k\times k}$ is a Haar-random orthogonal cross-block mixer, and $\mathbf{I}_h \in \mathbb{R}^{h\times h}$ is the identity matrix. The $\mathbf{Q}_k \otimes \mathbf{I}_h$ stage mixes corresponding coordinates across all k blocks, directly redistributing the residual block statistics left by RPBH. When $k = 1$, K-RPBH reduces to RPBH.

Proposition 1 (Cross-block variance averaging). *Let $x \in \mathbb{R}^d$ be a unit vector with $d = kh$, and let $M_j = \|(\mathbf{P}x)^{(j)}\|_2^2$ denote the energy of the j-th permuted block, with $\sum_{j=1}^{k} M_j = 1$. For the K-RPBH output $\hat{z}$, let $\hat{z}_\ell^{(r)}$ denote the ℓ-th coordinate of output block r. Conditioned on $\mathbf{P}$ and $\mathbf{Q}_k$,*

$$\mathrm{Var}_{\mathbf{D}}\left(\hat{z}_\ell^{(r)} \mid \mathbf{P}, \mathbf{Q}_k\right) = \frac{1}{h}\sum_{j=1}^{k} Q_{rj}^2 M_j, \tag{5}$$

where the variance is taken over the independent Rademacher sign matrices $\mathbf{D}_1, \ldots, \mathbf{D}_k$. If $\mathbf{Q}_k$ is Haar-random orthogonal, then

$$\mathbb{E}_{\mathbf{Q}_k}\left[\mathrm{Var}_{\mathbf{D}}\left(\hat{z}_\ell^{(r)} \mid \mathbf{P}, \mathbf{Q}_k\right)\right] = \frac{1}{d}. \tag{6}$$

The proof is provided in Appendix A. Figure 3 empirically validates this effect on representative FFN down-projection activations from Dream-7B (Ye et al., 2025) and Nemotron-Labs-Diffusion (Fu et al., 2026). K-RPBH brings the rotated coordinate marginal substantially closer to the Gaussian target and reduces the block-to-block conditional-variance fluctuation left by RPBH. This improved distributional matching translates directly to lower quantization error while retaining the efficiency of a structured rotation. K-RPBH remains substantially cheaper than the Kronecker transforms used in QuIP (Chee et al., 2023), and requires fewer transform operations than QuaRot (Ashkboos et al., 2024) on supported dimensions, as quantified in Figure 3(c,f).

4.3 Calibration-Free Weight–Activation Quantization

The predictable coordinate distribution produced by K-RPBH enables SoloQ to quantize weights and activations without calibration data. Consider a linear layer $y = \mathbf{W}x$, where $\mathbf{W} \in \mathbb{R}^{m\times d}$ and $x \in \mathbb{R}^d$. Following prior rotation-based quantization (Ashkboos et al., 2024; Lee et al., 2026a), SoloQ rotates both operands into the same K-RPBH basis and applies a quantizer $Q(\cdot)$:

$$\mathbf{W}' = \mathbf{W}\mathbf{\Pi}^\top, \qquad x' = \mathbf{\Pi}x, \qquad Q(\mathbf{W}')Q(x') \approx \mathbf{W}'x' = \mathbf{W}x, \tag{7}$$

where $\mathbf{\Pi}$ is the K-RPBH transform in Equation 4. The weight rotation is performed offline, while the activation rotation is applied online. SoloQ then supports either distribution-matched codebook quantization or hardware-native NVFP4 on the shared rotated representation.

Distribution-matched codebooks (SOLOQ-C). For codebook quantization, SOLOQ-C first computes scales from the rotated operands and normalizes them before quantization. Let w'_i denote the i-th row of $\mathbf{W}'$. We define

$$s_x = \|x'\|_2, \qquad s_{W,i} = \|w'_i\|_2, \qquad \tilde{x} = \frac{x'}{s_x}, \qquad \tilde{w}_i = \frac{w'_i}{s_{W,i}}. \tag{8}$$

The normalized coordinates are then quantized with distribution-matched codebooks,

$$\hat{x}'_{\mathrm{CB}} = s_x Q_{\mathrm{Lloyd}}(\tilde{x}), \qquad \hat{w}'_{i,\mathrm{CB}} = s_{W,i} Q_{\mathrm{E8P}}(\tilde{w}_i), \tag{9}$$

where Q_{Lloyd} is a Lloyd–Max scalar quantizer for activations (Zandieh et al., 2026), and Q_{E8P} is an E8P-based lattice quantizer for weights (Tseng et al., 2024), with residual stages for higher bit-widths. We use different quantizers because weights are quantized offline, where E8P search is affordable, whereas activations require lightweight runtime quantization. Both are derived from the predictable post-rotation distribution without calibration data or activation statistics.

Hardware-native NVFP4 (SOLOQ-N). SOLOQ also supports NVFP4 on the rotated representation, using the fixed E2M1 format with dynamic tensor- and block-wise scaling. For a rotated weight or activation tensor z, we compute the global amplitude $a_{\mathrm{g}} = \max_i |z_i|$ and block amplitude $a_j = \max_{i \in \mathcal{G}_j} |z_i|$ over groups of 16 coordinates. The corresponding scales are

$$s_{\mathrm{g}} = f_{\mathrm{g}}(a_{\mathrm{g}}), \qquad s_j = f_{\mathrm{E4M3}}(a_j), \tag{10}$$

and the quantized reconstruction is

$$\hat{z}^{(j)}_{\mathrm{NVFP4}} = s_{\mathrm{g}} s_j \, Q_{\mathrm{E2M1}}\left(\frac{z^{(j)}}{s_{\mathrm{g}} s_j}\right), \tag{11}$$

where Q_{E2M1} maps values to the representable E2M1 FP4 set. All scales are derived directly from z, without calibration data, pre-collected activation statistics, or learned parameters.

Thus, SOLOQ supports two calibration-free paths: SOLOQ-C with distribution-matched codebooks and SOLOQ-N with hardware-native NVFP4, both using the predictable rotated representation.

4.4 SCALE-ONLY BIAS CORRECTION

Low-bit quantization can distort both the direction and magnitude of a representation, particularly with centroid-based quantizers such as Lloyd–Max and, to a lesser extent, finite-precision formats such as NVFP4. Related scalar corrections have been used for unbiased compression in DRIVE (Vargaftik et al., 2021) and EDEN (Vargaftik et al., 2022), and for unbiased inner-product estimation in RaBitQ (Gao & Long, 2024). TurboQuant (Zandieh et al., 2026) instead uses a residual QJL sketch, requiring auxiliary information and a separate correction path. SOLOQ uses a scale-only bias correction to restore the projection along the original vector. Let $c_{\mathrm{bc}}(z)$ denote the corresponding bias-correction scale for a vector z. We define

$$c_{\mathrm{bc}}(z) = \frac{\|z\|_2^2}{\langle q(z), z\rangle}, \qquad \langle c_{\mathrm{bc}}(z) q(z), z\rangle = \|z\|_2^2. \tag{12}$$

Thus, the component along the original vector is exactly restored. Equivalently, decompose $q(z) = \alpha z + e_\perp$, where $\alpha = \langle q(z), z\rangle / \|z\|_2^2$ and $e_\perp \perp z$ denotes the orthogonal quantization error. Then $c_{\mathrm{bc}}(z) q(z) = z + e_\perp/\alpha$. Unlike residual-based corrections, c_{bc} requires no auxiliary payload or additional quantized bits. It is absorbed into the existing per-token activation or per-row weight scale, leaving the underlying low-bit linear operation unchanged.

4.5 BLOCK-DIFFUSION KV-CACHE QUANTIZATION

Full-sequence dLLMs such as LLaDA (Nie et al., 2026; Zhu et al., 2026) and Dream (Ye et al., 2025) recompute KV states throughout denoising and therefore do not maintain a persistent KV-cache. In contrast, block-diffusion models (Bie et al., 2026; Fu et al., 2026; Wu et al., 2026a) reuse the KV-cache of previously committed blocks, causing the persistent cache to grow with context length. As shown in Figure 1(a), this growth creates a significant memory bottleneck at long context lengths.

SOLOQ quantizes the KV-cache only when a block is committed, as illustrated in Figure 2(c). The active denoising block remains in full precision, avoiding repeated quantization across denoising steps. Keys use KIVI-style per-channel asymmetric quantization (Liu et al., 2024), while values follow the SOLOQ activation path with K-RPBH and either Lloyd–Max or NVFP4 quantization. This commit-time design avoids repeatedly quantizing the KV states of the actively denoised block while compressing only persistent states. The overall algorithm of SOLOQ is shown in Appendix B.

Table 1: W4A4 results on full-sequence diffusion language models. Values are task accuracy (%) and their average. SoloQ-C and SoloQ-N denote the codebook and NVFP4 variants, respectively. **Bold** and underlined indicate the best and second-best quantized results within each model.

Model	Method	Calibration-Free	Truth.	ARC-C	Hella.	Wino.	PIQA	MMLU	C-EVAL	Human.	GSM8K	Avg.
LLaDA-Base-8B	FP16	-	47.45	44.03	54.09	74.82	74.81	64.15	69.99	31.71	69.52	58.95
	RTN	✓	40.45	41.83	45.40	64.72	67.95	49.26	57.95	14.02	16.56	44.23
	AWQ	✗	40.87	42.92	46.14	66.88	69.43	51.22	58.43	20.10	36.88	48.09
	QuaRot+GPTQ	✗	42.53	44.20	49.76	69.85	70.75	55.96	56.32	25.33	44.57	51.03
	DLLMQuant+	✗	41.53	43.44	46.51	67.87	70.12	51.72	59.38	22.13	40.66	49.26
	DLLMQuant++	✗	43.53	44.18	51.00	71.85	73.94	57.77	61.22	28.92	56.25	54.29
	STaR-Quant	✗	**49.12**	44.23	52.75	72.92	73.85	62.95	64.56	**35.98**	57.29	57.07
	SoloQ-C	✓	45.22	43.52	52.94	74.51	74.16	64.11	67.53	29.88	**67.78**	**57.74**
	SoloQ-N	✓	45.33	**45.22**	**53.13**	**74.59**	**74.48**	**64.55**	**67.98**	25.61	67.48	57.60
LLaDA-1.5-8B	FP16	-	47.20	87.46	75.02	72.93	74.86	65.47	66.05	50.61	83.47	69.23
	RTN	✓	39.51	81.77	56.82	65.27	66.91	48.88	58.96	23.44	36.56	53.12
	AWQ	✗	40.96	82.22	66.84	67.93	69.32	51.12	60.03	30.07	57.95	58.49
	QuaRot+GPTQ	✗	43.21	84.23	65.34	69.55	70.17	56.23	57.66	37.33	65.86	61.06
	DLLMQuant+	✗	42.14	83.38	70.09	68.69	70.22	51.63	61.28	32.44	59.55	59.94
	DLLMQuant++	✗	43.87	84.18	69.20	71.74	73.58	57.27	60.04	44.58	74.33	64.31
	STaR-Quant	✗	**48.77**	85.12	72.43	**72.09**	73.36	63.29	64.34	46.53	76.45	66.93
	SoloQ-C	✓	48.10	**86.44**	73.01	71.59	74.65	64.60	64.71	46.95	80.29	67.82
	SoloQ-N	✓	48.09	86.10	**73.86**	71.67	**74.76**	**64.70**	**65.38**	**50.00**	**81.80**	**68.48**
Dream-7B†	FP16	-	43.96	59.98	73.73	74.43	75.46	71.85	62.48	59.15	77.79	66.54
	RTN	✓	41.25	54.34	58.32	64.52	66.26	51.51	49.89	28.90	30.82	49.53
	AWQ	✗	43.66	57.82	65.57	67.13	68.96	55.50	53.27	33.14	48.98	54.89
	QuaRot+GPTQ	✗	47.58	58.18	67.13	70.05	70.36	**69.50**	53.19	34.48	59.20	58.85
	DLLMQuant+	✗	44.14	58.38	66.93	68.66	69.53	57.63	54.21	35.12	51.14	56.19
	DLLMQuant++	✗	47.86	59.43	70.14	71.54	72.09	**69.50**	55.89	44.50	66.17	61.90
	STaR-Quant	✗	**48.72**	**59.76**	71.32	**72.98**	74.34	69.37	**58.79**	47.43	69.57	**63.59**
	SoloQ-C	✓	41.99	55.89	71.62	71.35	**74.76**	68.86	56.09	45.12	**72.86**	62.06
	SoloQ-N	✓	42.95	56.06	**72.05**	72.22	74.54	68.97	54.98	**50.61**	70.36	62.53

† TruthfulQA results for DLLMQuant/STaR-Quant are not reproducible (not open-sourced). Their FP is 49.76 vs. 43.96 in our setup.

Table 2: W4A4 and W4A4KV4 results on block-diffusion language models. Values are task accuracy (%) and average. SoloQ-C and SoloQ-N denote the codebook and NVFP4 variants. **Bold** and underlined mark the best and second-best quantized results per model and precision. Cb-free denotes whether the method is calibration-free, and N/A denotes an inapplicable method.

Model	Precision	Method	Cb-free	Human-B	Human-P	MBPP-B	MBPP-P	GSM8K	MATH	IFEval	MMLU	GPQA	Avg.
Fast-dLLM v2-7B	FP16	–	–	65.85	59.76	61.90	52.65	84.15	60.30	61.18	66.54	30.80	60.35
	W4A4	RTN	✓	0.00	0.00	0.00	0.00	0.23	0.00	7.58	24.73	27.46	6.67
		AWQ	✗	0.00	0.00	0.00	0.00	0.00	0.00	9.06	24.98	25.45	6.61
		QuaRot+GPTQ	✗	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
		SoloQ-C	✓	58.54	53.05	**60.05**	**50.53**	80.74	**54.86**	**59.89**	**64.73**	30.58	**57.00**
		SoloQ-N	✓	**59.10**	**56.10**	57.40	48.90	**82.64**	54.42	55.27	63.43	**33.71**	56.77
	W4A4KV4	SoloQ-C	✓	58.54	52.44	57.67	**51.32**	**79.98**	**54.06**	**56.38**	**64.73**	30.58	56.19
		SoloQ-N	✓	**59.15**	**55.49**	**57.94**	49.21	79.91	53.78	55.08	63.43	**31.92**	**56.21**
Nemotron-Labs Diffusion-8B	FP16	–	–	84.76	78.66	82.28	69.31	92.95	85.10	97.27	78.17	39.51	78.67
	W4A4	RTN	✓	0.00	0.00	0.00	0.00	0.00	0.00	11.70	0.00	0.00	1.30
		AWQ	✗	0.00	0.00	0.00	0.00	0.00	0.00	11.53	0.00	0.00	1.28
		QuaRot+GPTQ	✗	68.30	64.60	69.58	58.99	79.76	65.70	53.15	59.27	19.20	59.39
		SoloQ-C	✓	75.61	70.73	75.66	63.49	89.92	81.10	**62.35**	72.84	**35.94**	69.74
		SoloQ-N	✓	**76.22**	**72.56**	**78.84**	**67.46**	**91.21**	**81.12**	60.85	**74.17**	34.38	**70.76**
	W4A4KV4	SoloQ-C	✓	73.17	69.51	77.78	65.61	91.28	80.62	**65.48**	**73.98**	**34.15**	70.18
		SoloQ-N	✓	**75.00**	**72.56**	**80.16**	**67.99**	**91.66**	**81.10**	64.11	73.93	33.04	**71.06**

5 Experiments

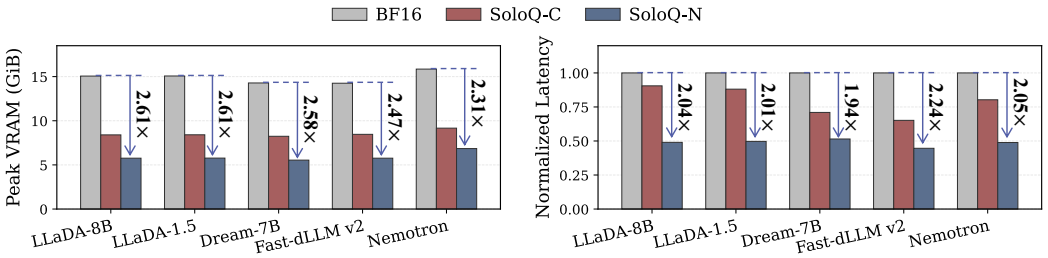
5.1 Experimental Setup

Models and evaluation. We evaluate SoloQ on both full-sequence and block-diffusion language models. For full-sequence dLLMs, we use LLaDA-8B-Base (Nie et al., 2026), LLaDA-1.5 (Zhu et al., 2026), and Dream-7B (Ye et al., 2025). For block-diffusion models, we evaluate Fast-dLLM v2 (Wu et al., 2026a) and Nemotron-Labs-Diffusion (Fu et al., 2026). All accuracy evaluations are conducted on NVIDIA H200 GPUs. We preserve each model's official architecture-specific evaluation stack rather than enforcing a single evaluator across all models. Detailed benchmark configurations are provided in Appendix C.

Baselines. For full-sequence dLLMs, we compare against standard LLM PTQ baselines including RTN, AWQ (Lin et al., 2024), QuaRot (Ashkboos et al., 2024), and GPTQ (Frantar et al., 2022), as well as dLLM-specific methods, DLLMQuant (Xu & Yang, 2025) and STaR-Quant (Yan et al., 2026). For block-diffusion models, where prior quantization results are limited, we compare against RTN, AWQ, and QuaRot under the same evaluation protocol. The additional comparison with Fair-Calib (Huang et al., 2026) is presented in Appendix D.1.

5.2 Full-Sequence Diffusion Language Models

Table 1 reports W4A4 results on LLaDA-Base (Nie et al., 2026), LLaDA-1.5 (Zhu et al., 2026), and Dream-7B (Ye et al., 2025). SoloQ outperforms existing quantized baselines without calibration



(a) Peak inference VRAM (b) Normalized end-to-end latency

Figure 4: **End-to-end efficiency of SoloQ.** (a) Peak inference VRAM and (b) normalized generation latency across five dLLMs. SoloQ-C and SoloQ-N are evaluated on H200 and RTX Pro 6000 GPUs, respectively, against their corresponding BF16 baselines.

Table 3: Rotation ablation under SoloQ-N, comparing task accuracy and rotation latency. Lat. denotes latency. N/A indicates dimensions unsupported by QuaRot.

Rotation	LLaDA-1.5		Fast-dLLM v2	
	Hella./MMLU (%)	Lat. (ms)	Human+/GPQA (%)	Lat. (ms)
QuIP	73.29/63.17	0.086	55.40/30.58	10.495
QuaRot	73.41/63.19	0.025	N/A	N/A
Haar	73.79/63.51	0.559	**57.90/33.48**	98.135
RPBH	73.08/63.48	0.031	53.00/32.81	2.447
K-RPBH	**73.86/64.70**	0.048	56.10/33.71	5.779

Table 4: Component ablation of K-RPBH on Nemotron. P denotes randomized permutation and Q_k cross-block mixing. Lower KL and energy imbalance indicate better distribution matching and global mixing.

Variant	P	Q_k	KL ↓	Energy Imbalance ↓
Block-RHT	✗	✗	0.11396	67.89%
Block-RHT + P	✓	✗	0.04683	45.64%
Block-RHT + Q_k	✗	✓	0.00682	36.24%
K-RPBH (ours)	✓	✓	**0.00413**	**22.12%**

data. SoloQ-C achieves 57.74 average accuracy on LLaDA-Base, improving over DLLMQuant++ by 3.45 points, while SoloQ-N reaches 68.48 on LLaDA-1.5, a 4.17-point gain. On Dream-7B, SoloQ-N achieves 62.53, versus 61.90 for DLLMQuant++. Overall, both variants provide robust W4A4 quantization across generation, reasoning, and knowledge tasks on full-sequence dLLMs.

5.3 Block-Diffusion Language Models

To assess quantization in block-diffusion settings, we evaluate W4A4 and W4A4KV4 on Fast-dLLM v2 (Wu et al., 2026a) and Nemotron-Labs-Diffusion (Fu et al., 2026), with results summarized in Table 2. Unlike full-sequence dLLMs, block-diffusion models maintain persistent KV states, making KV-cache quantization an additional efficiency opportunity. Most block-diffusion benchmarks are generation-based rather than likelihood-based, which can make them more sensitive to quantization error. As a result, RTN and AWQ collapse under W4A4 on both models, while QuaRot is not applicable to Fast-dLLM v2 because its rotation cannot support the model dimensions. Across both models, SoloQ maintains strong accuracy under W4A4 and W4A4KV4, demonstrating effective calibration-free quantization of weights, activations, and persistent KV states. Appendix D.4 analyzes attention-sink (Xiao et al., 2024; Rulli et al., 2026) behavior under KV-cache quantization.

5.4 Memory and Latency Efficiency

We measure the end-to-end inference efficiency of SoloQ using optimized GPU kernel implementations. SoloQ-C uses 8-bit Tensor Core kernels on NVIDIA H200 GPUs, while SoloQ-N uses native NVFP4 kernels on an NVIDIA RTX Pro 6000. As shown in Figure 4, SoloQ-N achieves a $2.31\times$–$2.61\times$ reduction in peak memory across five dLLMs, primarily through 4-bit weight storage and, for block-diffusion models, KV-cache compression. It also achieves $1.94\times$–$2.24\times$ end-to-end speedup over BF16 across the evaluated workloads. For Nemotron, whose EOS trajectory changes after quantization, we compare the same BF16-recorded sequence of 75 forward passes and obtain a $2.05\times$ speedup. We further implement SoloQ on an FPGA, as detailed in Appendix E.

6 Analysis

6.1 Rotation Structure Comparison

We first compare alternative rotations, QuIP (Chee et al., 2023), QuaRot (Ashkboos et al., 2024), Haar (Zandieh et al., 2026), and RPBH (Lee et al., 2026a), under the same SoloQ-N NVFP4 setting, as shown in Table 3. Rotation latency is measured for a single activation-rotation invocation

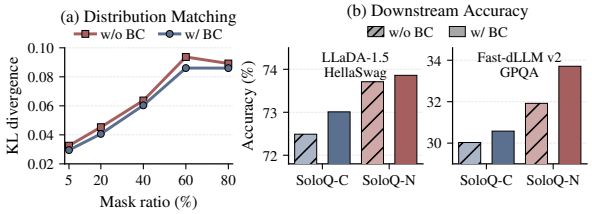


Figure 5: Effect of scale-only bias correction (BC). (a) KL divergence between full-precision and quantized logits across token mask ratios on LLaDA-1.5. (b) Downstream accuracy with and without BC.

Table 5: Comparison of 2-bit KV-cache quantization methods on Fast-dLLM v2 with weights and activations kept in FP16. Only the KV-cache is quantized, and BPE denotes its bit-per-entry.

Method	BPE	GSM8K (%)	IFEval (%)	Avg. (%)
FP16	16.00	84.15	61.18	72.67
KIVI	3.00	59.29	41.22	50.26
OScar	3.03	52.39	40.85	46.62
TurboQuant	3.18	0.30	7.76	4.03
SoloQ-C	3.00	**60.96**	**45.66**	**53.31**
SoloQ-N	3.00	**60.65**	**46.40**	**53.53**

on an NVIDIA H200 under each model's native evaluation workload. Dense Haar provides strong accuracy but incurs substantially higher cost. K-RPBH retains comparable accuracy while reducing rotation latency from 98.135 ms to 5.779 ms on Fast-dLLM v2, and from 0.559 ms to 0.048 ms on LLaDA-1.5. Compared with RPBH, K-RPBH consistently improves downstream accuracy, showing that cross-block mixing provides a better accuracy–efficiency trade-off.

6.2 K-RPBH Component Ablation

We next isolate the contributions of the randomized permutation P and cross-block mixer Q_k in K-RPBH (Eq. 4), as shown in Table 4. For a token representation $x \in \mathbb{R}^d$ partitioned into k blocks of size $h = d/k$, we define the energy of block j after permutation as $E_j(x) = \|(Px)^{(j)}\|_2^2$, whose uniform target is $\|x\|_2^2/k$. We measure block-energy imbalance as the median absolute relative deviation from this target, $\mathrm{Median}_{x,j}\left|kE_j(x)/\|x\|_2^2 - 1\right|$. For K-RPBH, the corresponding block energies are computed after cross-block mixing by Q_k. Lower imbalance indicates more uniform energy across blocks, while lower KL divergence indicates a closer match to the target post-rotation marginal. Starting from Block-RHT, adding P reduces KL from 0.114 to 0.0468 and energy imbalance from 67.89% to 45.64%. Adding only Q_k is more effective at redistribution, reducing KL to 0.0068 and imbalance to 36.24%. Combining both further improves the metrics to 0.0041 and 22.12%, respectively, confirming their complementary benefits.

6.3 Effect of Scale-Only Bias Correction

We further study the effect of the scale-only bias correction in Figure 5. Across all input-token masking ratios on LLaDA-1.5, the correction reduces the KL divergence between the full-precision and quantized logit distributions, indicating improved distribution matching. This translates to higher downstream accuracy for both SoloQ-C and SoloQ-N: HellaSwag improves from 72.49/73.71 to 73.01/73.86 on LLaDA-1.5, while GPQA improves from 30.03/31.92 to 30.58/33.71 on Fast-dLLM v2. Overall, the correction improves quantization fidelity with negligible overhead.

6.4 KV-Cache Quantization Comparison

Finally, we compare SoloQ with existing KV-cache quantization methods using 2-bit KV quantization at approximately 3 bits-per-entry (BPE), including scale and metadata overhead, in Table 5, while keeping weights and activations in FP16. SoloQ-C and SoloQ-N achieve average accuracies of 53.31 and 53.53, outperforming KIVI (Liu et al., 2024), OScar (Su et al., 2026), and TurboQuant (Zandieh et al., 2026). SoloQ-C performs best on GSM8K, while SoloQ-N achieves the highest IFEval accuracy, highlighting their complementary strengths. These results show that SoloQ effectively compresses the KV cache while preserving downstream accuracy.

7 Conclusion

We present SoloQ, a calibration-free quantization framework for diffusion language models. By transforming weights and activations into a predictable rotated representation, SoloQ enables data-independent low-bit quantization without calibration samples or pre-collected activation statistics. The framework supports both distribution-matched codebooks and hardware-native NVFP4, while commit-time KV-cache quantization extends the approach to block-diffusion models. Across full-sequence and block-diffusion dLLMs, SoloQ achieves robust W4A4 quantization and remains competitive with or outperforms existing calibration-based baselines across a broad range of language benchmarks. These results establish predictable representation shaping as a general foundation for efficient low-bit dLLM inference. Limitations are discussed in Appendix G.

AI USE STATEMENT

In this work, generative AI tools were used to aid and polish the writing, support the retrieval and discovery of related work, draft portions of the manuscript, and assist in proving mathematical claims, including suggesting intermediate derivation steps and refining proof structure. The authors checked all retrieved references against the original sources and independently reviewed and verified all AI-assisted mathematical arguments. All AI-generated text was revised and approved by the authors. The authors take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We provide detailed experimental settings, model-specific evaluation protocols, quantization procedures, ablation settings, and hardware configurations in Appendix C and the subsequent appendices to support reproducibility. We have aimed to report all implementation and evaluation details needed to reproduce the results as transparently as possible. Upon publication, we will release the code, configuration files, and evaluation scripts used in this work.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Marianne Arriola, Aaron Gokaslan, Justin Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, volume 2025, pp. 50726–50753, 2025.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021a.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021b.

Fan Bao, Shen Nie, Kaiwen Xue, Yue Cao, Chongxuan Li, Hang Su, and Jun Zhu. All are worth words: A vit backbone for diffusion models. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22669–22679. IEEE, 2023.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Ling Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, Jianfeng Tan, Lanning Wei, Ji-Rong Wen, Yipeng Xing, Xiaolu Zhang, Junbo Zhao, Da Zheng, Jun Zhou, Junlin Zhou, Zhanchao Zhou, Liwang Zhu, and Yihong Zhuang. Llada2.0: Scaling up diffusion language models to 100b, 2025. URL https://arxiv.org/abs/2512.15745.

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, Zhuocheng Gong, Yanmei Gu, Jian Guan, Kaiyuan Guan, Hongliang He, Zenan Huang, Juyong Jiang, Zhonghui Jiang, Zhenzhong Lan, Chengxi Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Yuan Lu, Yuxin Ma, Xingyu Mou, Zhenxuan Pan, Kaida Qiu, Yuji Ren, Jianfeng Tan, Yiding Tian, Zian Wang, Lanning Wei, Tao Wu, Yipeng Xing, Wentao Ye, Liangyu Zha, Tianze Zhang, Xiaolu Zhang, Junbo Zhao, Da Zheng, Hao Zhong, Wanli Zhong, Jun Zhou, Junlin Zhou, Liwang Zhu, Muzhi Zhu, and Yihong Zhuang. Llada2.1: Speeding up text diffusion via token editing, 2026. URL https://arxiv.org/abs/2602.08676.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. *Advances in neural information processing systems*, 36:4396–4429, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

OpenCompass Contributors. Opencompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass, 2023.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*, 2024.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Yonggan Fu, Lexington Whalen, Abhinav Garg, Chengyue Wu, Maksim Khadkevich, Nicolai Oswald, Enze Xie, Daniel Egert, Sharath Turuvekere Sreenivas, Shizhe Diao, et al. Nemotron-labs-diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding. *arXiv preprint arXiv:2607.05722*, 2026.

Jianyang Gao and Cheng Long. Rabitq: Quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. *Proceedings of the ACM on Management of Data*, 2(3):1–27, 2024.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Insu Han, Praneeth Kacham, Amin Karbasi, Vahab Mirrokni, and Amir Zandieh. Polarquant: Quantizing kv caches with polar transformation. *arXiv preprint arXiv:2502.02617*, 2025.

Zhengfu He, Tianxiang Sun, Qiong Tang, Kuanning Wang, Xuan-Jing Huang, and Xipeng Qiu. Diffusionbert: Improving generative masked language models with diffusion models. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 4521–4534, 2023.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Xing Hu, Yuan Cheng, Dawei Yang, Zukang Xu, Zhihang Yuan, Jiangyong Yu, Chen Xu, Zhe Jiang, and Sifan Zhou. Ostquant: Refining large language model quantization with orthogonal and scaling transformations for better distribution fitting. *arXiv preprint arXiv:2501.13987*, 2025.

Feice Huang, Zuliang Han, Xing Zhou, Yihuang Chen, Lifei Zhu, and Haoqian Wang. Convrot: Rotation-based plug-and-play 4-bit quantization for diffusion transformers. *arXiv preprint arXiv:2512.03673*, 2025.

Haoyu Huang, Linlin Yang, Sheng Xu, Boyu Liu, Guodong Guo, Zhongqian Fu, Hang Zhou, and Baochang Zhang. Fair-calib: Frontier-aware instability-reweighted calibration for post-training quantization of diffusion large language models. *arXiv preprint arXiv:2606.06547*, 2026.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Yao Fu, et al. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *Advances in neural information processing systems*, 36:62991–63010, 2023.

Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, et al. Hunyuanvideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2024.

Donghyun Lee, Jitesh Chavan, Duy Nguyen, Sam Huang, Liming Jiang, Priyadarshini Panda, Timo Mertens, and Saurabh Shukla. Orbitquant: Data-agnostic quantization for image and video diffusion transformers. *arXiv preprint arXiv:2607.02461*, 2026a.

Donghyun Lee, Yuhang Li, Ruokai Yin, and Priyadarshini Panda. Kronq: Llm quantization via kronecker-factored hessian. *arXiv preprint arXiv:2607.07964*, 2026b.

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. Diffusion-lm improves controllable text generation. *Advances in neural information processing systems*, 35: 4328–4343, 2022.

Yuhang Li, Ruokai Yin, Donghyun Lee, Shiting Xiao, and Priyadarshini Panda. Gptaq: Efficient finetuning-free quantization for asymmetric calibration. *arXiv preprint arXiv:2504.02692*, 2025.

Zhiteng Li, Hanxuan Li, Junyi Wu, Kai Liu, Haotong Qin, Linghe Kong, Guihai Chen, Yulun Zhang, and Xiaokang Yang. Dvd-quant: data-free video diffusion transformers quantization. In *International Conference on Learning Representations*, volume 2026, pp. 35463–35475, 2026.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 3214–3252, 2022.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: Llm quantization with learned rotations. In *International Conference on Learning Representations*, volume 2025, pp. 92009–92032, 2025.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *Advances in Neural Information Processing Systems*, 38:50608–50646, 2026.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182. IEEE, 2023.

Fitsum Reda, John Kamalu, Roger Waleffe, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-twotower: Diffusion language modeling with pretrained autoregressive context. *arXiv preprint arXiv:2606.26493*, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Maximo Eduardo Rulli, Simone Petruzzi, Edoardo Michielon, Fabrizio Silvestri, Simone Scardapane, and Alessio Devoto. Attention sinks in diffusion language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 43244–43256, 2026.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Gao Peng, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. In *International Conference on Learning Representations*, volume 2024, pp. 45472–45496, 2024.

Zunhai Su, Rui Yang, Chao Zhang, Yaxiu Liu, Yifan Zhang, Wei Wu, Jing Xiong, Dayou Du, Xialie Zhuang, Yulei Qian, et al. Oscar: The occam's razor for extreme kv cache quantization in llms and beyond. *arXiv preprint arXiv:2605.19660*, 2026.

DiffusionGemma Team, Adrien Ali Taïga, James Assiene, Daniele Calandriello, Rahma Chaabouni, João Gante, Tamara von Glehn, Nate Keating, Chris Knutsen, Martin Kukla, et al. Diffusiongemma technical report. *arXiv preprint arXiv:2608.00146*, 2026.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *Proceedings of machine learning research*, 235:48630, 2024.

Shay Vargaftik, Ran Ben-Basat, Amit Portnoy, Gal Mendelson, Yaniv Ben-Itzhak, and Michael Mitzenmacher. Drive: One-bit distributed mean estimation. *Advances in Neural Information Processing Systems*, 34:362–377, 2021.

Shay Vargaftik, Ran Ben Basat, Amit Portnoy, Gal Mendelson, Yaniv Ben Itzhak, and Michael Mitzenmacher. Eden: Communication-efficient and robust distributed mean estimation for federated learning. In *International Conference on Machine Learning*, pp. 21984–22014. PMLR, 2022.

Xiuying Wei, Yunchen Zhang, Yuhang Li, Xiangguo Zhang, Ruihao Gong, Jinyang Guo, and Xianglong Liu. Outlier suppression+: Accurate quantization of large language models by equivalent and effective shifting and scaling. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1648–1665, 2023.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng-ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, et al. Qwen-image technical report. *arXiv preprint arXiv:2508.02324*, 2025.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm. In *International Conference on Learning Representations*, volume 2026, pp. 128353–128370, 2026a.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. In *International Conference on Learning Representations*, volume 2026, pp. 57027–57051, 2026b.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pp. 38087–38099. PMLR, 2023.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

Chen Xu and Dawei Yang. Dllmquant: Quantizing diffusion-based large language models. *arXiv preprint arXiv:2508.14090*, 2025.

Xin Yan, Aqiang Wang, Zhenglin Wan, Xingrui Yu, and Ivor Tsang. Star-quant: State-time consistent post-training quantization for diffusion large language models. *arXiv preprint arXiv:2606.04945*, 2026.

Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. Cogvideox: Text-to-video diffusion models with an expert transformer. In *International Conference on Learning Representations*, volume 2025, pp. 83048–83077, 2025.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. Turboquant: Online vector quantization with near-optimal distortion rate. In *International Conference on Learning Representations*, volume 2026, pp. 56418–56439, 2026.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Tianao Zhang, Zhiteng Li, Xianglong Yan, Haotong Qin, Yong Guo, and Yulun Zhang. Quant-dllm: Post-training extreme low-bit quantization for diffusion large language models. In *International Conference on Learning Representations*, volume 2026, pp. 79492–79503, 2026.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11425–11460, 2026.

A Proof of Proposition 1

Let $y = \mathbf{P}x$ and partition it into k blocks $y^{(1)}, \dots, y^{(k)} \in \mathbb{R}^h$. Define the energy of the j-th block as

$$M_j = \left\| y^{(j)} \right\|_2^2, \qquad \sum_{j=1}^{k} M_j = 1, \tag{13}$$

where the equality follows from $\|x\|_2 = 1$ and the orthogonality of the permutation matrix $\mathbf{P}$.

Before the cross-block mixing, the output of the j-th RPBH block is

$$z^{(j)} = \mathbf{H}_h \mathbf{D}_j y^{(j)}. \tag{14}$$

For its ℓ-th coordinate,

$$z_\ell^{(j)} = \sum_{m=1}^{h} H_{\ell m} D_{j,mm} y_m^{(j)}. \tag{15}$$

Since the diagonal entries of $\mathbf{D}_j$ are independent Rademacher variables with zero mean and unit variance, and the normalized Walsh–Hadamard matrix satisfies $H_{\ell m}^2 = 1/h$, we obtain

$$\mathbb{E}_{\mathbf{D}_j}\left[z_\ell^{(j)}\right] = 0, \qquad \mathrm{Var}_{\mathbf{D}_j}\left(z_\ell^{(j)}\right) = \frac{M_j}{h}. \tag{16}$$

K-RPBH subsequently applies $\mathbf{Q}_k \otimes \mathbf{I}_h$, which mixes the same within-block coordinate across the k blocks. Thus, the ℓ-th coordinate of output block r is

$$\hat{z}_\ell^{(r)} = \sum_{j=1}^{k} Q_{rj} z_\ell^{(j)}. \tag{17}$$

Conditioned on $\mathbf{P}$ and $\mathbf{Q}_k$, the variables $z_\ell^{(1)}, \dots, z_\ell^{(k)}$ are independent because the sign matrices $\mathbf{D}_1, \dots, \mathbf{D}_k$ are sampled independently. Therefore,

$$\begin{aligned} \mathrm{Var}_{\mathbf{D}}\left(\hat{z}_\ell^{(r)} \mid \mathbf{P}, \mathbf{Q}_k\right) &= \sum_{j=1}^{k} Q_{rj}^2 \,\mathrm{Var}_{\mathbf{D}_j}\left(z_\ell^{(j)}\right) \\ &= \frac{1}{h} \sum_{j=1}^{k} Q_{rj}^2 M_j, \end{aligned} \tag{18}$$

which proves Equation 5.

Finally, each row of a Haar-random orthogonal matrix $\mathbf{Q}_k$ is uniformly distributed on the unit sphere $\mathbb{S}^{k-1}$, and hence

$$\mathbb{E}_{\mathbf{Q}_k}\left[Q_{rj}^2\right] = \frac{1}{k}. \tag{19}$$

Taking expectation over $\mathbf{Q}_k$ in Equation 18 gives

$$\begin{aligned} \mathbb{E}_{\mathbf{Q}_k}\left[\mathrm{Var}_{\mathbf{D}}\left(\hat{z}_\ell^{(r)} \mid \mathbf{P}, \mathbf{Q}_k\right)\right] &= \frac{1}{h} \sum_{j=1}^{k} \mathbb{E}_{\mathbf{Q}_k}\left[Q_{rj}^2\right] M_j \\ &= \frac{1}{hk} \sum_{j=1}^{k} M_j = \frac{1}{hk} = \frac{1}{d}, \end{aligned} \tag{20}$$

where we use $\sum_j M_j = 1$ and $d = kh$. This proves Equation 6. □

B ALGORITHM

For compactness, let $\mathrm{SQ}^{W}_{\nu,b}(\cdot)$ and $\mathrm{SQ}^{A}_{\nu,b}(\cdot)$ denote the SOLOQ weight and activation quantization operators, respectively, for variant $\nu \in \{\mathrm{C}, \mathrm{N}\}$ and bit-width b. Each operator includes its respective scaling, quantization, and scale-only bias correction. For committed keys, let $\mathrm{AsymPC}_b(\cdot)$ denote b-bit per-channel asymmetric quantization along the token dimension. Here, $\mathcal{D}$ denotes the set of rotation dimensions used by the target layers and KV values, N the number of tokens, B the active block, and d_v the value-vector dimension.

Algorithm 1 SOLOQ offline weight and online activation/KV-cache quantization

Require: Transformer $\mathcal{T}$, target layers $\mathcal{L}$, rotation dimensions $\mathcal{D}$, variant $\nu \in \{\mathrm{C}, \mathrm{N}\}$, bit-widths $(b_w, b_a, b_{\mathrm{kv}})$

▷ **Offline**
for $d \in \mathcal{D}$ **do**
  $\Pi_d \leftarrow \text{K-RPBH}(d)$
end for
for each $\mathbf{W} \in \mathcal{L}$ with input dimension d **do**
  $\mathbf{W}' \leftarrow \mathbf{W}\Pi_d^\top$
  $\widehat{\mathbf{W}}' \leftarrow \mathrm{SQ}^{W}_{\nu,b_w}(\mathbf{W}')$
  Replace $\mathbf{W}$ with $\widehat{\mathbf{W}}'$ in $\mathcal{T}$
end for
▷ **Online activation quantization** for $\mathbf{X} \in \mathbb{R}^{N\times d}$
$\mathbf{X}' \leftarrow \mathbf{X}\Pi_d^\top$
$\widehat{\mathbf{X}}' \leftarrow \mathrm{SQ}^{A}_{\nu,b_a}(\mathbf{X}')$
▷ **Commit-time KV quantization** for block-diffusion models
Keep the active KV block $(\mathbf{K}_B, \mathbf{V}_B)$ in full precision
On commit, $\widehat{\mathbf{K}}_B \leftarrow \mathrm{AsymPC}_{b_{\mathrm{kv}}}(\mathbf{K}_B)$
On commit, $\widehat{\mathbf{V}}_B \leftarrow \mathrm{SQ}^{A}_{\nu,b_{\mathrm{kv}}}\left(\mathbf{V}_B\Pi_{d_v}^\top\right)$
Append $(\widehat{\mathbf{K}}_B, \widehat{\mathbf{V}}_B)$ to the persistent KV-cache
return Quantized $\mathcal{T}$ and persistent KV-cache

C EVALUATION DETAILS

We retain each model's official evaluation stack rather than enforcing a unified harness, since identically named benchmarks can differ in prompting, inference, and grading. Within each model, FP and quantized runs use identical prompts, few-shot settings, batch sizes, generation parameters, and graders. For full-sequence models, we evaluate TruthfulQA (Lin et al., 2022), ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), PIQA (Bisk et al., 2020), MMLU (Hendrycks et al., 2020), C-Eval (Huang et al., 2023), HumanEval (Chen et al., 2021), and GSM8K (Cobbe et al., 2021). For block-diffusion models, Human-B/P and MBPP-B/P denote the Base and Plus variants of HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021b), evaluated using EvalPlus (Liu et al., 2023). We additionally evaluate GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), IFEval (Zhou et al., 2023), MMLU (Hendrycks et al., 2020), and GPQA (Rein et al., 2023).

C.1 LLADA-8B-BASE

For LLaDA-8B-Base, we follow the official LM Evaluation Harness setup (Gao et al., 2024). TruthfulQA-MC2, ARC-Challenge, HellaSwag, PIQA, and WinoGrande use 128 Monte Carlo masking samples, while MMLU and C-EVAL use one. TruthfulQA uses CFG 2.0, ARC-Challenge/HellaSwag/PIQA use CFG 0.5, and the remaining tasks use CFG 0. WinoGrande, MMLU, and C-EVAL are 5-shot; the other likelihood tasks are zero-shot. All likelihood tasks report plain accuracy. GSM8K follows the official 5-shot setup with generation length, diffusion steps, and block length of 1,024 and strict-match scoring. HumanEval is zero-shot with the same generation configuration and reports pass@1.

C.2 LLaDA-1.5

LLaDA-1.5 is evaluated with its official OpenCompass stack (Contributors, 2023) for ARC-Challenge, HellaSwag, MMLU, GSM8K, and HumanEval. ARC-Challenge uses zero-shot generation with length/steps/block size 512; HellaSwag and MMLU use generation length/steps/block size 3, with MMLU evaluated 5-shot. GSM8K uses four in-prompt demonstrations with 256/256/16, and HumanEval uses 512/512/32. All generation tasks use greedy decoding and low-confidence remasking. ARC-Challenge and HellaSwag use first-option accuracy, MMLU reports the macro-average across 57 subjects, GSM8K uses answer extraction, and HumanEval reports pass@1. For compatibility with prior quantization work, TruthfulQA-MC2, WinoGrande, PIQA, and C-EVAL are additionally evaluated using the LLaDA-8B likelihood protocol; these columns are supplementary and not part of the official LLaDA-1.5 generation suite.

C.3 Dream-v0-Base-7B

Dream-v0-Base-7B is evaluated with its native evaluator, including the Monte Carlo ELBO estimator, autoregressive logit shift, and BOS token. All likelihood tasks use 128 Monte Carlo samples. ARC-Challenge, HellaSwag, PIQA, and TruthfulQA-MC2 are zero-shot, while WinoGrande, MMLU, and C-EVAL are 5-shot. We use batch size 32 for ARC-Challenge/HellaSwag/PIQA/WinoGrande/TruthfulQA, 16 for MMLU, and 1 for C-EVAL; the reduced MMLU batch size is applied consistently to FP and quantized runs. ARC-Challenge, HellaSwag, and PIQA report normalized accuracy, while the remaining likelihood tasks report accuracy. GSM8K uses 8-shot CoT prompting with 256 generated tokens and 256 diffusion steps; HumanEval is zero-shot with 512 generated tokens and 512 diffusion steps. TruthfulQA and C-EVAL are used only for controlled FP-to-quantized comparisons.

C.4 Fast-dLLM v2

For Fast-dLLM v2, evaluation follows the official NVLabs adapter. GSM8K, Minerva MATH, and IFEval are zero-shot with batch size 32; MMLU is zero-shot and GPQA Main 5-shot with batch size 1. Generation uses block size 32, sub-block size 8, threshold 1, greedy decoding, and up to 2,048 new tokens. MMLU and GPQA are evaluated by likelihood rather than generation. GSM8K uses flexible answer extraction, Minerva MATH uses symbolic-equivalence rescoring, and IFEval reports prompt-level strict accuracy. HumanEval and MBPP use EvalPlus with batch size 32, reporting both base and Plus pass@1 after code sanitization.

C.5 Nemotron-Labs-Diffusion-8B

Nemotron-Labs-Diffusion-8B is evaluated with the official inference implementation and NeMo-Skills stack. All tasks, including MMLU and GPQA, are evaluated through generation with block length 32, threshold 0.9, temperature 0, batch size 1, and up to 8,192 generated tokens. Thinking mode and EOS early stopping are disabled. Since the Nemotron paper does not specify block length or EOS policy for its main table, we treat this as a controlled operating point rather than an exact reconstruction. For the cross-model table, we evaluate GSM8K, Hendrycks MATH, IFEval, MMLU, GPQA Main, HumanEval, and MBPP. Mathematical tasks use the NeMo-Skills answer evaluator, MMLU and GPQA use generation-based multiple-choice scoring, HumanEval and MBPP use EvalPlus, and IFEval is averaged over its four strict/loose instruction-following metrics. MATH-500 and GPQA Diamond are reported only in comparisons matching the Nemotron paper's benchmark configuration.

D Additional Analysis

D.1 Comparisons with FAIR-Calib

FAIR-Calib (Huang et al., 2026) is a calibration-based PTQ method for dLLMs that prioritizes fragile token states near the write frontier, where quantization errors can alter token decisions before they are committed. It first probes a full-precision teacher to estimate a frontier-aware position prior, then performs off-policy layer-wise calibration with a reweighted hidden-state reconstruction loss. Since FAIR-Calib uses an evaluation suite different from our main full-sequence setting, we report a separate W4A4 comparison on the five shared benchmarks: TruthfulQA, WinoGrande, PIQA, MMLU, and GSM8K. As shown in Table 6, SoloQ achieves higher average accuracy on both LLaDA-Base and Dream-7B without calibration data.

Table 6: Comparison with FAIR-Calib under W4A4 quantization on the five benchmarks shared by both evaluation settings. Results for FP16, RTN, QuaRot, FlatQuant, and FAIR-Calib are taken from FAIR-Calib, while the remaining baselines and SOLOQ results follow our evaluation. Values are task accuracy (%) and their average. **Bold** and underlined indicate the best and second-best quantized results within each model.

Model	Method	Calibration-Free	Truth.	Wino.	PIQA	MMLU	GSM8K	Avg.
LLaDA-Base	FP16	-	45.30	73.64	74.84	65.80	68.92	65.70
	RTN	✓	38.80	61.80	69.26	51.05	35.03	51.19
	AWQ	✗	40.87	66.88	69.43	51.22	36.88	53.06
	QuaRot	✗	43.35	72.22	73.61	62.04	57.39	61.72
	FlatQuant	✗	42.90	72.16	74.16	63.80	57.24	62.05
	DLLMQuant+	✗	41.53	67.87	70.12	51.72	40.66	54.38
	DLLMQuant++	✗	43.53	71.85	73.94	57.77	56.25	60.67
	FAIR-Calib	✗	43.91	72.93	**74.92**	64.11	64.75	64.12
	SOLOQ-C	✓	45.22	74.51	74.16	64.11	**67.78**	65.16
	SOLOQ-N	✓	**45.33**	**74.59**	74.48	**64.55**	67.48	**65.29**
Dream-7B	FP16	-	44.17	73.32	75.41	71.36	76.19	68.09
	RTN	✓	41.42	55.64	61.70	36.94	16.83	42.51
	AWQ	✗	43.66	67.13	68.96	55.50	48.98	56.85
	QuaRot	✗	40.40	63.46	72.52	59.74	49.51	57.13
	FlatQuant	✗	43.23	65.98	71.65	64.96	60.20	61.20
	DLLMQuant+	✗	44.14	68.66	69.53	57.63	51.14	58.22
	DLLMQuant++	✗	**47.86**	71.54	72.09	**69.50**	66.17	65.43
	FAIR-Calib	✗	43.40	69.53	73.07	66.91	66.64	63.91
	SOLOQ-C	✓	41.99	71.35	**74.76**	68.86	**72.86**	**65.96**
	SOLOQ-N	✓	42.95	**72.22**	74.54	68.97	70.36	65.81

D.2 UNIFORM INT4 QUANTIZATION

To evaluate whether SOLOQ depends on specialized non-uniform quantizers, we additionally replace the codebook and NVFP4 quantizers with standard uniform INT4 quantization while keeping the K-RPBH rotation and the remaining pipeline unchanged. We denote this variant as SOLOQ-I. Tables 7 and 8 compare SOLOQ-I with SOLOQ-C, SOLOQ-N, and existing W4A4 baselines. SOLOQ-I remains competitive across all three architectures, achieving average accuracies of 56.98 on LLaDA-Base, 67.44 on LLaDA-1.5, and 55.32 on Fast-dLLM v2. The degradation relative to SOLOQ-C/N is modest, indicating that the predictable representation produced by K-RPBH also supports conventional uniform INT4 quantization without calibration.

Table 7: Comparison with uniform INT4 quantization on full-sequence diffusion language models. SOLOQ-C, SOLOQ-N, and SOLOQ-I denote the codebook, NVFP4, and uniform INT4 variants, respectively. All SOLOQ variants use the same unfolded K-RPBH topology. Values are task accuracy (%) and their average.

Model	Method	Truth.	ARC-C	Hella.	Wino.	PIQA	MMLU	C-EVAL	Human.	GSM8K	Avg.
LLaDA-Base	FP16	47.45	44.03	54.09	74.82	74.81	64.15	69.99	31.71	69.52	58.95
	RTN	40.45	41.83	45.40	64.72	67.95	49.26	57.95	14.02	16.56	44.23
	AWQ	40.87	42.92	46.14	66.88	69.43	51.22	58.43	20.10	36.88	48.09
	QuaRot	42.53	44.20	49.76	69.85	70.75	55.96	56.32	25.33	44.57	51.03
	DLLMQuant+	41.53	43.44	46.51	67.87	70.12	51.72	59.38	22.13	40.66	49.26
	DLLMQuant++	43.53	44.18	51.00	71.85	73.94	57.77	61.22	28.92	56.25	54.29
	STaR-Quant	49.12	44.23	52.75	72.92	73.85	62.95	64.56	35.98	57.29	57.07
	SOLOQ-C	45.22	43.52	52.94	74.51	74.16	64.11	67.53	29.88	67.78	57.74
	SOLOQ-N	45.33	45.22	53.13	74.59	74.48	64.55	67.98	25.61	67.48	57.60
	SOLOQ-I	45.56	44.20	52.21	73.32	73.94	63.77	67.24	26.22	66.34	56.98
LLaDA-1.5	FP16	47.20	87.46	75.02	72.93	74.86	65.47	66.05	50.61	83.47	69.23
	RTN	39.51	81.77	56.82	65.27	66.91	48.88	58.96	23.44	36.56	53.12
	AWQ	40.96	82.22	66.84	67.93	69.32	51.12	60.03	30.07	57.95	58.49
	QuaRot	43.21	84.23	65.34	69.55	70.17	56.23	57.66	37.33	65.86	61.06
	DLLMQuant+	42.14	83.38	70.09	68.69	70.22	51.63	61.28	32.44	59.55	59.94
	DLLMQuant++	43.87	84.18	69.20	71.74	73.58	57.27	60.04	44.58	74.33	64.31
	STaR-Quant	48.77	85.12	72.43	72.09	73.36	63.29	64.34	46.53	76.45	66.93
	SOLOQ-C	48.10	86.44	73.01	71.59	74.65	64.60	64.71	46.95	80.29	67.82
	SOLOQ-N	48.09	86.10	73.86	71.67	74.76	64.70	65.38	50.00	81.80	68.48
	SOLOQ-I	48.68	87.46	72.43	71.27	74.65	64.12	65.23	43.90	79.23	67.44

Table 8: Comparison with uniform INT4 quantization on Fast-dLLM v2. SOLOQ-C, SOLOQ-N, and SOLOQ-I denote the codebook, NVFP4, and uniform INT4 variants, respectively. All SOLOQ variants use the same unfolded K-RPBH topology. Values are task accuracy (%) and their average. N/A indicates that the method is not applicable to the architecture.

Model	Method	Human-B	Human-P	MBPP-B	MBPP-P	GSM8K	MATH	IFEval	MMLU	GPQA	Avg.
Fast-dLLM v2	FP16	65.85	59.76	61.90	52.65	84.15	60.30	61.18	66.54	30.80	60.35
	RTN	0.00	0.00	0.00	0.00	0.23	0.00	7.58	24.73	27.46	6.67
	AWQ	0.00	0.00	0.00	0.00	0.00	0.00	9.06	24.98	25.45	6.61
	QuaRot	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	SOLOQ-C	58.54	53.05	60.05	50.53	80.74	54.86	59.89	64.73	30.58	57.00
	SOLOQ-N	59.10	56.10	57.40	48.90	82.64	54.42	55.27	63.43	33.71	56.77
	SOLOQ-I	60.37	56.10	54.50	48.94	79.30	52.74	54.16	61.41	30.36	55.32

D.3 FOLDED ROTATION FOR EFFICIENT DEPLOYMENT

Online activation rotation introduces additional inference overhead, motivating us to examine whether the residual-stream rotation can be folded into model weights offline. This is difficult in image or video diffusion transformers such as those targeted by OrbitQuant (Lee et al., 2026a), which commonly employ adaptive normalization such as AdaLN (Peebles & Xie, 2023). Let $x \in \mathbb{R}^d$ be a hidden state, $W \in \mathbb{R}^{m\times d}$ a linear weight, and $\Pi \in \mathbb{R}^{d\times d}$ an orthogonal K-RPBH rotation. We use $\mathcal{N}(\cdot)$ to denote RMS normalization without its learned channel-wise gain. AdaLN can be written as

$$\mathrm{AdaLN}(x; c) = \mathrm{diag}(\gamma(c))\,\mathcal{N}(x) + \beta(c), \tag{21}$$

where the scale $\gamma(c)$ and shift $\beta(c)$ depend on the diffusion condition c. Because these parameters vary at runtime, the residual-stream rotation cannot generally be absorbed into fixed adjacent weights and remains online. In contrast, the dLLMs considered in SOLOQ use a static RMSNorm gain γ. Since orthogonal rotations preserve the RMS norm, the full-precision operation can be equivalently written as

$$W\,\mathrm{diag}(\gamma)\mathcal{N}(x) = \underbrace{W\,\mathrm{diag}(\gamma)\Pi^\top}_{W_{\mathrm{fold}}}\mathcal{N}(\Pi x). \tag{22}$$

Thus, the RMSNorm gain and residual-stream rotation can be folded into W_{fold} offline, following the QuaRot-style topology and reducing online rotation overhead. However, this equivalence does not generally survive quantization:

$$Q_W(W\Pi^\top)Q_A(\Pi\,\mathrm{diag}(\gamma)\mathcal{N}(x)) \neq Q_W(W_{\mathrm{fold}})Q_A(\mathcal{N}(\Pi x))\,. \tag{23}$$

Tables 9 and 10 quantify this trade-off. Although folding reduces online rotation cost, it introduces model-dependent accuracy degradation, with a relatively modest drop on LLaDA and a substantially larger drop on Fast-dLLM v2. We therefore view folding as a deployment-time efficiency–accuracy trade-off rather than an accuracy-preserving transformation under low-bit quantization.

Table 9: Comparison of unfolded and folded SOLOQ under W4A4 quantization on full-sequence diffusion language models. *Unfold* explicitly applies the activation-side rotations, whereas *Fold* absorbs foldable residual-stream rotations into adjacent weights offline.

Model	Method	Topology	Truth.	ARC-C	Hella.	Wino.	PIQA	MMLU	C-EVAL	Human.	GSM8K	Avg.
LLaDA-Base	FP16	–	47.45	44.03	54.09	74.82	74.81	64.15	69.99	31.71	69.52	58.95
	RTN	–	40.45	41.83	45.40	64.72	67.95	49.26	57.95	14.02	16.56	44.23
	AWQ	–	40.87	42.92	46.14	66.88	69.43	51.22	58.43	20.10	36.88	48.09
	QuaRot	–	42.53	44.20	49.76	69.85	70.75	55.96	56.32	25.33	44.57	51.03
	DLLMQuant++	–	43.53	44.18	51.00	71.85	73.94	57.77	61.22	28.92	56.25	54.29
	STaR-Quant	–	49.12	44.23	52.75	72.92	73.85	62.95	64.56	35.98	57.29	57.07
	SOLOQ-C	Unfold	45.22	43.52	52.94	74.51	74.16	64.11	67.53	29.88	67.78	57.74
	SOLOQ-C	Fold	43.72	43.17	51.00	71.98	73.45	62.27	66.05	27.44	64.06	55.96
	SOLOQ-N	Unfold	45.33	45.22	53.13	74.59	74.48	64.55	67.98	25.61	67.48	57.60
	SOLOQ-N	Fold	41.60	43.52	51.10	72.30	73.56	61.84	66.72	24.39	62.70	55.30
LLaDA-1.5	FP16	–	47.20	87.46	75.02	72.93	74.86	65.47	66.05	50.61	83.47	69.23
	RTN	–	39.51	81.77	56.82	65.27	66.91	48.88	58.96	23.44	36.56	53.12
	AWQ	–	40.96	82.22	66.84	67.93	69.32	51.12	60.03	30.07	57.95	58.49
	QuaRot	–	43.21	84.23	65.34	69.55	70.17	56.23	57.66	37.33	65.86	61.06
	DLLMQuant++	–	43.87	84.18	69.20	71.74	73.58	57.27	60.04	44.58	74.33	64.31
	STaR-Quant	–	48.77	85.12	72.43	72.09	73.36	63.29	64.34	46.53	76.45	66.93
	SOLOQ-C	Unfold	48.10	86.44	73.01	71.59	74.65	64.60	64.71	46.95	80.29	67.82
	SOLOQ-C	Fold	48.58	86.78	71.80	70.56	74.65	63.47	65.53	46.34	76.95	67.18
	SOLOQ-N	Unfold	48.09	86.10	73.86	71.67	74.76	64.70	65.38	50.00	81.80	68.48
	SOLOQ-N	Fold	49.16	85.76	72.11	72.53	74.97	63.75	65.23	46.95	79.23	67.74

Table 10: Comparison of unfolded and folded SoloQ under W4A4 quantization on Fast-dLLM v2. *Unfold* explicitly applies activation-side rotations, whereas *Fold* absorbs foldable residual-stream rotations into adjacent weights offline. Values are task accuracy (%) and their average. N/A indicates that the method is not applicable to the corresponding architecture.

Model	Method	Topology	Human-B	Human-P	MBPP-B	MBPP-P	GSM8K	MATH	IFEval	MMLU	GPQA	Avg.
Fast-dLLM v2	FP16	–	65.85	59.76	61.90	52.65	84.15	60.30	61.18	66.54	30.80	60.35
	RTN	–	0.00	0.00	0.00	0.00	0.23	0.00	7.58	24.73	27.46	6.67
	AWQ	–	0.00	0.00	0.00	0.00	0.00	0.00	9.06	24.98	25.45	6.61
	QuaRot	–	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	SoloQ-C	Unfold	58.54	53.05	60.05	50.53	80.74	54.86	59.89	64.73	30.58	57.00
	SoloQ-C	Fold	51.22	47.56	52.38	44.97	77.56	49.38	51.20	56.70	27.46	50.94
	SoloQ-N	Unfold	59.10	56.10	57.40	48.90	82.64	54.42	55.27	63.43	33.71	56.77
	SoloQ-N	Fold	59.10	54.90	52.40	47.10	74.60	48.36	50.65	57.80	30.36	52.81

D.4 Attention sink in dLLM

Attention sinks, where a small number of token positions receive disproportionately large attention mass, have also been observed in diffusion language models (Rulli et al., 2026). Figure 6 shows that this behavior persists in block-diffusion models: Fast-dLLM v2 exhibits a pronounced sink at position 2, while Nemotron-Labs-Diffusion concentrates attention at position 0. Because such positions can play a disproportionate role in attention computation, aggressively quantizing their KV states may cause larger degradation than quantizing ordinary positions.

Table 11 shows that preserving the attention-sink position in full precision has little effect under W4A4KV4, suggesting that 4-bit KV quantization is sufficiently robust to the concentrated attention mass. In contrast, under the more aggressive W4A4KV2 setting, preserving the sink becomes beneficial. This indicates that sink-aware protection is mainly important at ultra-low KV precision, while standard 4-bit KV quantization can generally quantize the sink together with the remaining cache without noticeable degradation.

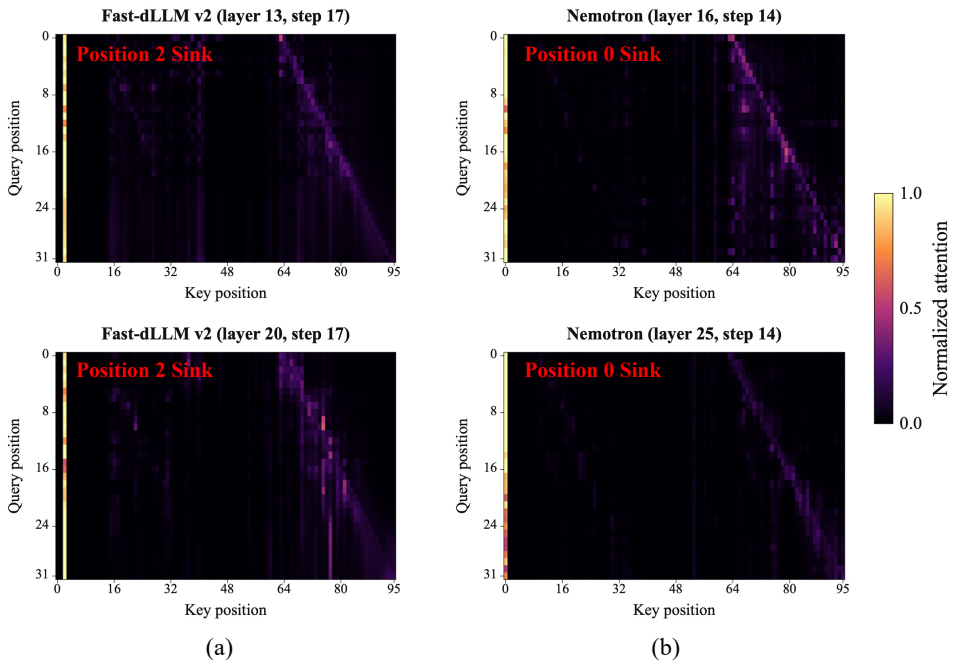
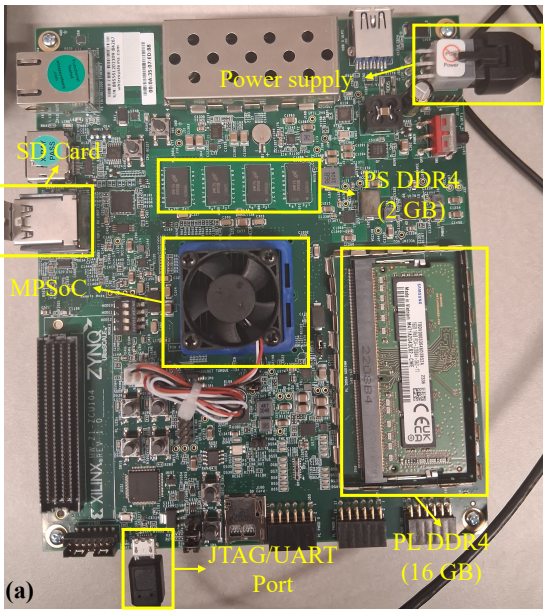


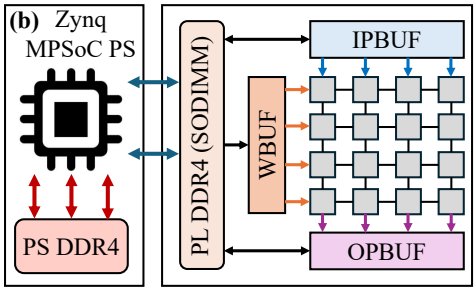
Figure 6: Attention-sink patterns in block-diffusion language models. (a) Fast-dLLM v2 exhibits a concentrated sink at position 2 across representative layers. (b) Nemotron-Labs-Diffusion exhibits a sink at position 0. Brighter colors indicate larger attention weights.

Table 11: Effect of preserving attention-sink KV states in full precision under aggressive KV-cache quantization. Sink FP indicates that the identified attention-sink position is kept in full precision while the remaining KV states are quantized.

Model	Precision	Sink FP	GSM8K	IFEval
Fast-dLLM v2	W4A4KV4	✓	80.59	56.75
		✗	79.98	56.38
	W4A4KV2	✓	55.80	41.22
		✗	50.57	39.93
Nemotron-Labs-Diffusion	W4A4KV4	✓	90.98	65.48
		✗	91.28	65.06
	W4A4KV2	✓	82.94	54.18
		✗	72.93	52.64

E FPGA IMPLEMENTATION





(c)

Parameter	Value
Array size	16 x 16
Accumulator width	64 bits
IPBUF, WBUF, OPBUF	1 MB each
Core clock	150 MHz
Memory interface clock	266.5 MHz
Peak DRAM bandwidth	4.8 GB/s

Figure 7: (a) Hardware setup on ZynQ Ultrascale+ ZCU104 FPGA board. (b) Micro-architecture of the hardware accelerator. (c) Hardware accelerator design parameters.

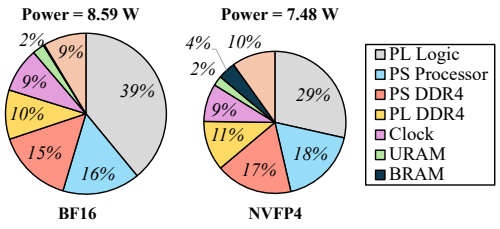


Figure 8: Power breakdown of BF16 and NVFP4 accelerators for LLaDA-Base.

Table 12: Throughput and energy efficiency of BF16 and NVFP4 accelerators on LLaDA-Base.

Metric	BF16	NVFP4
Tokens/sec	0.13 (1×)	**0.252 (1.94×)**
Tokens/kJ	15.13 (1×)	**40.90 (2.70×)**

We implement a systolic-array accelerator with custom BF16 and NVFP4 datapaths on a Zynq UltraScale+ ZCU104 FPGA and evaluate LLaDA-Base-8B for edge dLLM inference. Fig. 7(a) shows the hardware setup. The Zynq MPSoC consists of a processing system (PS) and programmable logic

(PL), each with access to external DDR4 memory. The model weights are distributed across the PS and PL DDR4 memories, while activations are stored in the PL DDR4.

As shown in Fig. 7(b), the systolic-array accelerator is implemented in the PL and receives execution instructions and a portion of the model weights from the PS. The systolic array comprises multiply-and-accumulate (MAC) units together with input (IP), weight (W), and output (OP) buffers to perform tiled matrix multiplication on data tiles fetched from the PL DDR4.

Fig. 7(c) summarizes the design specifications of the hardware accelerator. For the NVFP4 accelerator, we double the compute capacity because NVFP4 enables higher compute density than BF16. Fig. 8 shows the power breakdown of the accelerators during LLaDA-Base-8B inference, where the compute logic, DDR memories, PS processor, and clock generation and routing are the major contributors. The NVFP4 accelerator consumes 15% less power while providing twice the compute throughput of the BF16 design. As shown in Table 12, SoloQ's NVFP4 accelerator achieves a $\mathbf{1.94}\times$ end-to-end speedup and $\mathbf{2.70}\times$ higher energy efficiency than BF16 on LLaDA-Base-8B.

F Qualitative Generation Examples

We provide generations to illustrate how low-bit quantization affects model outputs beyond aggregate benchmark scores. The examples cover mathematical reasoning, code generation, and general instruction following, including both agreement and divergent outputs among FP16, SoloQ-C, and SoloQ-N. The mathematical reasoning examples are generated with Fast-dLLM v2-7B, while the code-generation examples use LLaDA-8B-Base. All variants use identical prompts and model-specific deterministic decoding configurations. These examples are selected for qualitative inspection only. All quantitative claims in the paper are based on the complete benchmark results.

Mathematical reasoning — Divergent outputs

Prompt. *A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts in total does it take?*

FP16 ✓

White fiber $= 2/2 = 1$ bolt. Total $= 2 + 1 = 3$ bolts. Final answer: $\boxed{3}$.

SoloQ-C ✓

$B = 2$, $W = B/2 = 1$, and $B + W = 2 + 1 = 3$. Final answer: $\boxed{3}$.

SoloQ-N ✓

White fiber $= (1/2) \times 2 = 1$ bolt. Total $= 2 + 1 = 3$. Final answer: $\boxed{3}$.

Code generation — Correct Outputs

Prompt. *Implement* `has_close_elements(numbers, threshold)`.

FP16 ✓

```
if len(numbers) < 2:
    return False
for i in range(len(numbers)):
    for j in range(i + 1, len(numbers)):
        if abs(numbers[i] - numbers[j]) < threshold:
            return True
return False
```

SOLOQ-C ✓

```
if len(numbers) < 2:
    return False
sorted_numbers = sorted(numbers)
for i in range(len(sorted_numbers) - 1):
    if abs(sorted_numbers[i] - sorted_numbers[i + 1]) < threshold:
        return True
return False
```

SOLOQ-N ✓

```
if len(numbers) < 2:
    return False
for i in range(len(numbers)):
    for j in range(i + 1, len(numbers)):
        if abs(numbers[i] - numbers[j]) < threshold:
            return True
return False
```

Mathematical reasoning — Incorrect Outputs

Prompt. *Josh decides to try flipping a house. He buys a house for $80,000 and then puts in $50,000 in repairs. This increased the value of the house by 150%. How much profit did he make?*

FP16 ✗

Computes $80{,}000 + 50{,}000 = 130{,}000$, then $130{,}000 - 50{,}000 = 80{,}000$. Final answer: $\boxed{80000}$.

SOLOQ-C ✗

Computes the new value as $80{,}000 + 1.5(80{,}000) = 200{,}000$, then $200{,}000 - 80{,}000 - 50{,}000 = 170{,}000$. Final answer: $\boxed{170000}$.

SOLOQ-N ✓

Computes total cost as $80{,}000 + 50{,}000 = 130{,}000$ and the new value as $80{,}000 + 120{,}000 = 200{,}000$, giving a profit of $200{,}000 - 130{,}000 = 70{,}000$. Final answer: $\boxed{70000}$.

General explanation — length-constrained instruction following

Prompt. *Explain to a non-specialist how AI models can be made to use less memory and respond faster. Describe a few common approaches, give a simple everyday example, and explain the trade-offs between speed, cost, and answer quality. Use clear language and keep your answer to about 250–300 words.*

FP16

Sure, I'd be happy to explain this in simple terms! AI models can be made to use less memory and respond faster through several approaches. Here are a few:

1. Model Compression: This involves reducing the size of the model without significantly impacting its performance. For example, you can remove less important connections in a neural network, or use techniques like quantization to reduce the precision of the weights. Imagine compress a JPEG image - you can reduce its size without losing much quality. In AI, this can help reduce memory usage and improve response times.

2. Efficient Model Architect: Some models are designed to be more efficient from the. For instance, MobileNet is a convolutional neural network designed for mobile devices. It uses fewer parameters and computations than traditional models, making it faster and less memory-intensive.

3. Hardware Optimization: This involves using hardware that's optimized for AI tasks, like GPUs or TPUs. These can perform complex calculations much faster than regular CPUs, reducing response times.

4. Batch Processing: This involves processing multiple inputs at once, rather than one by one. This can reduce the overhead of individual function calls and improve memory usage.

The trade-offs between speed, cost, and answer quality are important. For example, compressing a model can reduce memory usage and improve response times, but it might also reduce the accuracy of the model. Using an efficient model architecture can improve speed and cost, but it might require more expertise to design and implement. Hardware optimization can improve speed and cost, but it might require expensive hardware. Batch processing can improve speed and cost, but it might require more memory.

In conclusion, there are many ways to make AI models use less memory and respond faster, but each has its trade-offs. It's important to consider the specific requirements and constraints of the application when deciding which approach to use.

(312 words; 395 tokens; EOS)

. . . . . .

SOLOQ-C

Sure, I'd be happy to explain this in simple terms!

AI models can be made to use less memory and respond faster through several approaches. Here are a few:

1. Model Compression: This is like compress a big, complex book into a smaller, simpler one. You can do this by removing unnecessary parts, using shared layers, or applying techniques like pruning or quantization. For example, a model that originally used 1GB of memory might be compressed to 100MB, making it run faster and use less memory.

2. Efficient Architectures: These are like using a specialized tool for a specific job. For instance, MobileNet is designed for mobile devices with limited resources. It uses fewer parameters and computations than other models, making it faster and more efficient.

3. Hardware-Aware Training: This is like designing a racecar for a specific track. By training the model on the target hardware, you can optimize it for that hardware, improving its speed and efficiency.

There are some trade-offs to consider:

Speed: Fasteraller models generally run faster, but they might not be as accurate as larger models. For example, a smaller image model might take longer to train but could be faster during inference.

Cost: Smaller models can be cheaper to train and deploy, but they might require more computational resources to achieve similar accuracy.

Answer Quality: Smaller models might provide less detailed or accurate answers than larger models. For instance, a smaller language model might struggle to understand complex sentences or generate nuanced responses.

In conclusion, the key is to balance the need for speed and memory efficiency with the desire for accurate and detailed answers.

(274 words; 363 tokens; EOS)

. . . . . .

SOLOQ-N

Sure, I'd be happy to explain this in simple terms!

AI models can be made to use less memory and respond faster through several approaches. Here are a few common ones:

1. Model Compression: This involves reducing the size of the model by removing redundant weights or using more efficient representations. For example, you can use a technique called "pruning" to remove less important weights from the model, making it smaller and faster. Another technique is "quantization", where you reduce the precision of the weights, also making the model smaller and faster.

2. Knowledge Distillation: This is a process where a smaller model is trained to mimic the behavior of a larger model. For instance, you can train a simple model to copy the predictions of a complex model, but in a way that the simple model can make those predictions faster and with less memory.

3. Hardware Optimization: This involves optimizing the hardware to run the model more efficiently. For example, you can use a specialized chip like a GPU or a tensor processing unit (TPU) to speed the model up and reduce memory usage.

Let's consider a simple everyday example. Imagine you have a recipe for a complex dish that takes a long time to prepare and requires a lot of ingredients. You can make it simpler (model compression) by removing some of the ingredients (weights), make it smaller (knowledge distillation) by teaching a simpler recipe to mimic the complex one, or optimize the preparation process (hardware optimization) by using a more efficient kitchen.

However, there are trade-offs. Making the model simpler or smaller might reduce the quality (answer quality) of the predictions. For example, a simpler model might not capture as much complexity in the data, leading to less accurate predictions. Similarly, making the model faster might require more resources (cost) to run, or it might not be as accurate as a slower model. Therefore, it's important to balance speed, cost, and answer quality when optimizing AI models.

(331 words; 427 tokens; EOS)

The examples illustrate that low-bit perturbations can affect both task-level correctness and generation behavior. In the agreement cases, quantized models preserve the reasoning or functional

behavior of the full-precision model. In the divergent cases, small representation perturbations alter discrete token decisions, sometimes correcting an FP16 failure and sometimes introducing a new error. The general instruction-following example further shows that the overall structure and content of a response can remain similar across precisions while details such as wording, examples, and response length vary. These observations motivate evaluating quantization over complete benchmark suites rather than relying on individual generations.

G LIMITATIONS

SOLOQ introduces online activation rotation to transform representations into the K-RPBH basis. Although weight rotations are performed offline, activation-side rotations incur additional inference overhead. Folding these rotations into adjacent weights can reduce this cost, but, as shown in Appendix D.3, the folded topology may degrade accuracy under low-bit quantization, particularly for block-diffusion models. In addition, SOLOQ-C uses non-uniform codebooks whose centroids are represented losslessly in INT8 for efficient computation. Consequently, its low-bit representation cannot directly exploit native 4-bit Tensor Core execution and instead relies on 8-bit kernels, resulting in lower speedups than the hardware-native NVFP4 path of SOLOQ-N. Finally, the full latency benefit of SOLOQ-N depends on hardware support for native NVFP4 computation; on devices without such support, its practical acceleration may be reduced.