MUQUANT: MASK STATE ADAPTIVE DUAL-CENTER QUANTIZATION FOR DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Block-diffusion language models repeatedly process partially masked blocks during generation. Within a decoding step, masked and unmasked tokens can have distinct activation centers, making a shared quantization origin poorly suited to either population. Calibration on fully unmasked text also misses the activation mixtures encountered during denoising. MUQuant addresses these two sources of W4A4 error with a state-conditioned representation. MuZero estimates separate centers for masked and unmasked tokens, and MuCal calibrates weights on the reconstructed activations collected along denoising rollouts. MuEngine then computes the center and residual contributions in one packed four-bit multiplication, avoiding separate weight passes. Across three diffusion language-model families, this approach improves quantized accuracy while preserving the decoding procedure. On Nemotron-Labs-Diffusion-8B, the evaluated NVFP4 implementation achieves up to $3.24\times$ speedups over BF16.

1 INTRODUCTION

Diffusion language models (dLLMs) have emerged as a practical alternative to autoregressive generation (Nie et al., 2025; Ye et al., 2025a; Cheng et al., 2025). Rather than extending a response one token at a time, a dLLM repeatedly revisits a partially masked response or active block and unmasks several tokens over an iterative trajectory. The same linear layers are consequently evaluated many times, so even a modest reduction in the cost of one forward pass compounds over generation. Existing serving systems address this cost from a complementary direction. Fast-dLLM uses approximate KV caching and parallel commits (Wu et al., 2026), while dLLM-Cache reuses stable features across adjacent steps (Liu et al., 2026). Quantization instead lowers the cost of the forward passes that remain, without changing the sampler's update rule.

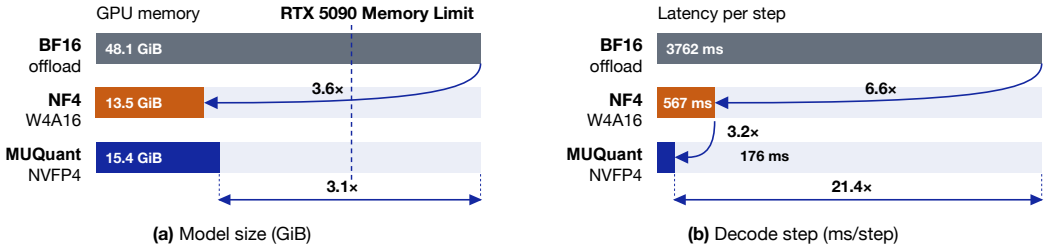


Figure 1: **DiffusionGemma-26B on RTX 5090.** MUQuant uses 15.4 GiB and runs $3.2\times$ faster per decode step than NF4 W4A16, at a cost of 1.9 GiB additional memory. BF16 requires offloading, so its latency includes data transfers.

Weight-only W4A16 reduces the traffic associated with model weights, but it leaves activations and arithmetic in high precision. W4A4 quantizes the activation operand as well, enabling native low-bit weight–activation multiplication as in Atom, COMET, and QuaRot (Zhao et al., 2024; Liu et al., 2024; Ashkboos et al., 2024). Realizing these savings in a dLLM requires a representation that remains accurate as the active block changes during denoising. In particular, partially masked

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.

decoding does not produce a single homogeneous activation distribution. Prior dLLM quantization accounts for distributions that evolve across denoising steps (Xu et al., 2026). Beyond this step-wise variation, we find an additional separation within each step: masked and unmasked tokens occupy different channel-wise centers (Figure 3). A symmetric four-bit quantizer must spend part of its already limited range on this offset, leaving fewer levels to model the variation that matters around either population. Per-token scaling adapts the grid spacing but does not remove this channel-wise offset. The sampler already identifies the two populations, suggesting that their shared offsets can be represented separately while reserving the quantization grid for token-level variation.

We propose MU(μ)Quant, a W4A4 quantization framework that captures the distinct activation distributions of masked and unmasked tokens using state-dependent mean (μ) centers. It comprises MuZero, MuCal, and MuEngine. MuZero computes these channel-wise means separately on each forward pass, then quantizes the centers and each token's residual from its reconstructed center. Tokens thus share their state-dependent offset without discarding its contribution to the output.

Weight calibration must account for this representation as well as the state mixtures absent from fully unmasked text. MuCal collects intermediate rollout states with their mask indicators and reconstructs their activations with MuZero before accumulating GPTQ statistics. Weight rounding therefore reflects the deployed state mixtures and activation rounding errors, without changing GPTQ's update procedure.

The remaining challenge is to retain this representation's accuracy without paying for two weight passes. MuEngine stacks the at-most-two center rows with the token residual rows for one packed 4-bit GEMM, then adds each token's center output to its residual output. Figure 2 illustrates the benefit under a weight-only traffic model without inter-GEMM cache reuse: with 64 token rows and two centers, packing raises the intensity from 33 to 66 MACs per fetched weight element. Both points remain on the bandwidth-limited slope, where avoiding a second weight fetch matters more than the higher W4A4 compute ceiling (Williams et al., 2009).

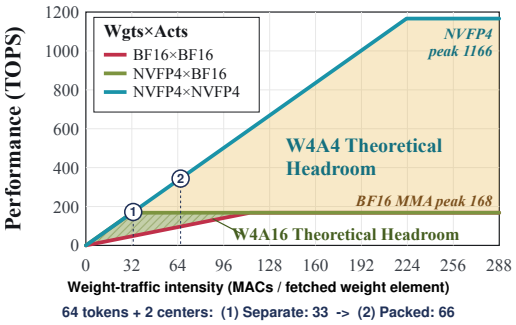


Figure 2: **Theoretical RTX 5090 roofline.** Separate (1) versus packed (2) W4A4. Markers indicate theoretical bounds, not measurements.

Contributions.

- MuZero removes the within-step masked/unmasked center gap with two online origins.
- MuCal fits GPTQ weights to rollout states after MuZero reconstruction.
- MuEngine executes both origins with one weight read and one packed 4-bit GEMM.

We evaluate accuracy on Nemotron-Labs-Diffusion-8B, SDAR-8B-64, and DiffusionGemma-26B (Tables 1 and 2). Figure 1 shows DiffusionGemma-26B deployment under memory constraints, while the other two models' efficiency measurements keep the models resident on the GPU.

2 Related Work

Post-training quantization. LLM.int8() isolates outlier features in higher precision (Dettmers et al., 2022), while ZeroQuant uses fine-grained weight and activation quantization (Yao et al., 2022). SmoothQuant rescales channels to redistribute quantization difficulty between activations and weights (Xiao et al., 2023). QuaRot and SpinQuant use rotations to reduce outliers (Ashkboos et al., 2024; Liu et al., 2025), while GPTQ uses activation statistics to compensate weight rounding (Frantar et al., 2023). Atom, COMET, and SVDQuant pair low-bit representations with efficient kernels (Zhao et al., 2024; Liu et al., 2024; Li et al., 2025). MUQuant addresses an additive offset conditioned on token state, and modifies calibration and execution to account for that offset.

Diffusion-LLM quantization. DLLMQuant (Xu et al., 2026) approaches dLLM quantization through temporal-mask sampling (TMAS), attention-value quantization (IA-AQ), and mask- and confidence-weighted Hessians (CGQ). MUQuant instead treats the within-step offset between token states as part of the linear-layer representation, then carries that representation through calibration and execution. MuZero quantizes residuals around two online centers, MuCal calibrates the reconstructed activations, and MuEngine packs both contributions into one GEMM. The distinction lies in representing and executing the state-dependent offset explicitly, rather than using token state only to guide calibration.

dLLM serving. Block Diffusion combines block-wise generation with parallel token sampling (Arriola et al., 2025). Fast-dLLM (Wu et al., 2026) and dLLM-Cache (Liu et al., 2026) reduce or reuse denoising work. MUQuant leaves that trajectory intact and lowers each remaining forward's linear cost.

3 Quantization Preliminaries

3.1 W4A4 quantization

Quantization replaces high-precision values with low-bit codes by scaling them onto a finite grid. Rounding selects a grid point, and restoring the scale gives an approximation to the original tensor:

$$\widehat{Z} = \mathrm{clip}(\mathrm{round}(Z/s_Z), -q_{\max}, q_{\max}), \quad s_Z = \frac{\max|Z|}{q_{\max}}, \quad Q_b(Z) = s_Z\widehat{Z}, \tag{1}$$

where Z is the input tensor, $\widehat{Z}$ its integer code, s_Z its scale, and $Q_b(Z)$ its reconstruction, with $q_{\max} = 2^{b-1} - 1$ for signed integers. Applying this approximation to both operands enables four-bit linear computation:

$$XW^\top \approx Q_4(X)Q_4(W)^\top = s_X s_W \widehat{X}\widehat{W}^\top. \tag{2}$$

where $X \in \mathbb{R}^{N\times C_i}$ contains N token activations and $W \in \mathbb{R}^{C_o\times C_i}$ contains the weights. Scale granularity is suppressed for clarity. Matching operand widths enables native low-bit multiplication, whereas mixed widths may require upcasting. At four bits, however, a small number of activation outliers can enlarge the range and coarsen the grid for the remaining values. SVDQuant shows that smoothing can be insufficient in diffusion models: it shifts activation outliers into the weights and absorbs the resulting weight outliers with a high-precision low-rank branch (Li et al., 2025). MUQuant instead targets a state-dependent offset shared by subsets of tokens. Equation 1 describes the INT4 case. For NVFP4, Q_4 instead denotes reconstruction from the floating-point codebook and its associated scales. The state-centered decomposition below applies to both formats.

3.2 GPTQ

Rounding weights independently ignores how their errors combine at the layer output. GPTQ instead chooses quantized weights to preserve that output on calibration data (Frantar et al., 2023):

$$\min_{\widehat{W}\in\mathcal{Q}} \|X\widehat{W}^\top - XW^\top\|_F^2, \qquad H = X^\top X + \lambda I. \tag{3}$$

where X contains calibration activations, W and $\widehat{W}$ are the original and quantized weights, and $\mathcal{Q}$ is the representable set. The Hessian approximation H uses damping $\lambda > 0$ and absorbs the common factor of two into its normalization. GPTQ implements OBQ updates with blockwise column processing and a Cholesky reformulation. Its weight rounding is determined by statistics of the calibration activations through H. Teacher-forced activations can therefore be mismatched when deployment repeatedly visits partially masked states, motivating MuCal in Section 4.2. The coordinate-wise update is deferred to Appendix A.

4 Method

MUQuant writes each activation as a state-dependent center plus a residual. We begin by defining that representation, then show how it changes the calibration statistics and how the resulting terms can still be executed in one packed multiplication. The denoising sampler already provides the token state, so the method does not require an additional predictor.

4.1 MuZero: two state-dependent origins

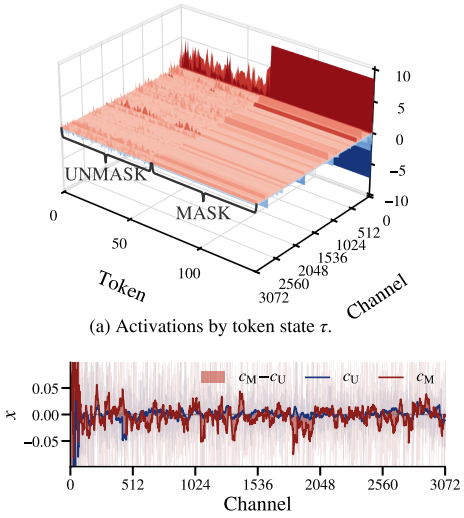
At an intermediate denoising step, some tokens are still masked while others have already been unmasked. Although the two groups pass through the same linear layer, they do not need to be centered at the same location. Figure 3 makes this visible in the first layer, where the two populations follow distinct channel-wise centers.

State-conditioned centers. Each token should be centered relative to others in the same state. We therefore average MASK and UNMASK activations separately, then assign each token its state's mean:

$$c_t^a = \frac{1}{|\mathcal{I}_t^a|} \sum_{i \in \mathcal{I}_t^a} X_t[i,:], \qquad C_t[i,:] = c_t^{s_t(i)}. \tag{4}$$

where $X_t \in \mathbb{R}^{N \times C_i}$ contains activations at step t, $s_t(i) \in \{\mathrm{M}, \mathrm{U}\}$ is token i's state, and $\mathcal{I}_t^a = \{i : s_t(i) = a\}$. For each nonempty state, c_t^a is its channel-wise mean and C_t assigns that mean to its tokens. Recomputing these centers on each forward pass follows the current distribution rather than a fixed offline offset.

Channel-wise smoothing applies one transformation to every token in a channel. When the two populations have different means, it cannot center both simultaneously. Conditioning the origin on token state removes this additive offset before per-token scaling is applied. The grouping is thus determined by the known mask indicator, rather than inferred from activation similarity.



(a) Activations by token state τ.

(b) State centres c_M, c_U; shading is their gap.

Figure 3: **Within-step activation separation.** MASK and UNMASK tokens have distinct channel-wise centers. The lower panel shows their means and the gap between them.

Quantized reconstruction. Rounding a center introduces an error that the residual can absorb if we subtract the quantized center first. We therefore select each token's MASK or UNMASK center, quantize it, and represent the remaining variation with a second four-bit term. Both contributions retain their dependence on token state and share the same weights, which allows the packed multiplication in Figure 5. This gives the reconstruction

$$R_t = X_t - Q_4(C_t), \qquad \widetilde{X}_t = Q_4(C_t) + Q_4(R_t). \tag{5}$$

where R_t is the residual after center quantization and $\widetilde{X}_t$ is the reconstructed activation. If the residual were exact, $Q_4(C_t) + R_t$ would recover X_t even when the center is rounded. With four-bit residuals, the remaining activation error is consequently $Q_4(R_t) - R_t$. The linear layer computes $\widetilde{X}_t Q_4(W)^\top$, so both terms use the same weight operand. This shared operand, together with the two-row center structure, is what makes the packed implementation in Section 4.3 possible.

4.2 MuCal: Modified GPTQ for the MUQuant operator

Weight calibration must reflect both the states visited during decoding and the activation representation used at inference. MuCal does this by collecting denoising rollouts and accumulating statistics after MuZero reconstruction. Once those statistics have been formed, the rounding procedure remains standard GPTQ. Figure 4 illustrates the activation mixtures absent from fully unmasked calibration text.

Rollout-state calibration set. Teacher-forced text shows the model only at its fully unmasked endpoint, even though deployment spends most of its work at intermediate decoding states. TMAS in DLLMQuant (Xu et al., 2026) addresses this mismatch by stratifying calibration over timesteps and mask ratios. MuCal instead records $\mathcal{D}_{\mathrm{roll}} = \{(X_t, s_t)\}$ along each calibration example's own

denoising trajectory (Figure 4) and preserves the mixture as it occurs. The resulting Hessian therefore reflects the MASK/UNMASK proportions encountered by the deployed sampler, rather than an artificially balanced population.

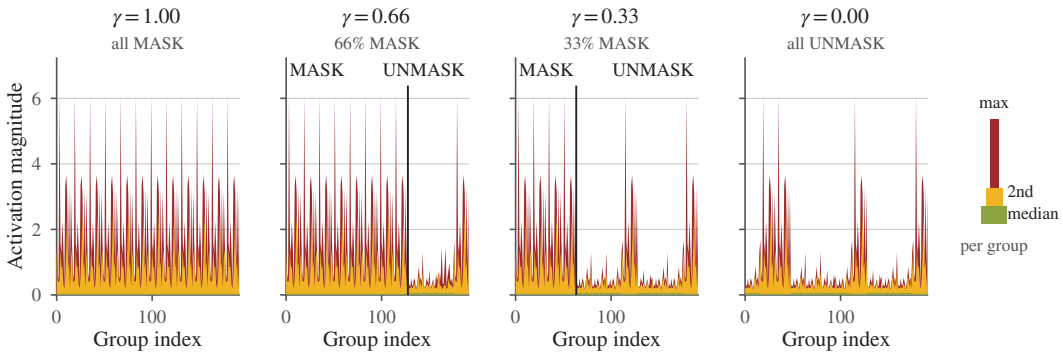


Figure 4: **Rollout-state coverage for MuCal.** Teacher-forced text covers only the fully unmasked endpoint, whereas rollout sampling covers deployed MASK/UNMASK mixtures.

Algorithm 1 MuCal: GPTQ with operator-matched rollout statistics

$\mathcal{D}_{\text{roll}} \leftarrow \text{DenoiseRollout}(\mathcal{D}_{\text{cal}})$ *// deployment states*
$C_t \leftarrow \text{StateCenters}(X_t, s_t), \quad \widetilde{X}_t \leftarrow Q_4(C_t) + Q_4(X_t - Q_4(C_t))$ *// reconstructed input*
$H \leftarrow \sum_{(X_t,s_t)\in\mathcal{D}_{\text{roll}}} \widetilde{X}_t^\top \widetilde{X}_t + \lambda I$ *// matched Hessian*
$Q \leftarrow 0_{d_{\text{row}}\times d_{\text{col}}}, \quad E \leftarrow 0_{d_{\text{row}}\times B}$ *// GPTQ buffers*
$U \leftarrow \text{Cholesky}(\text{inverse}(H))^\top$ *// upper-triangular factor*
for $i = 0, B, 2B, \ldots < d_{\text{col}}$ **do**
  $b \leftarrow \min(i + B, d_{\text{col}})$, **for** $j = i, \ldots, b-1$ **do**
    $Q_{:,j} \leftarrow \text{quant}(W_{:,j})$ *// quantize column*
    $E_{:,j-i} \leftarrow (W_{:,j} - Q_{:,j})/U_{jj}$ *// quantization error*
    $W_{:,j:b} \leftarrow W_{:,j:b} - E_{:,j-i}U_{j,j:b}$ *// update block*
  end for
  $W_{:,b:} \leftarrow W_{:,b:} - E_{:,0:(b-i)}U_{i:b,b:}$ *// update remainder*
end for
return $\text{Pack}(Q)$ *// packed output*

Operator-matched reconstruction. For each recorded state, MuCal reconstructs $\widetilde{X}_t$ with Equation 5 and accumulates H from this deployed representation rather than from the raw teacher-forced activation. It then evaluates both the full-precision and rounded weights against the same $\widetilde{X}_t$. In this way, the calibration objective isolates weight-rounding error after the activation representation has already been fixed. MuCal neither learns the state centers nor introduces a separate activation model.

More precisely, replacing X in Equation 3 gives the calibration objective

$$\min_{\widehat{W}\in\mathcal{Q}} \sum_t \left\| \widetilde{X}_t (\widehat{W} - W)^\top \right\|_F^2, \qquad H_{\text{cal}} = \sum_t \widetilde{X}_t^\top \widetilde{X}_t + \lambda I. \tag{6}$$

where $\widetilde{X}_t$ is the reconstructed rollout activation and H_{cal} collects its damped second moments. The objective measures weight-rounding error with the activation operator fixed, while MuZero addresses activation reconstruction error. Using raw X_t instead would weight rounding errors according to a representation that the deployed multiplication does not see.

Weight rounding. Once these statistics are formed, GPTQ processes weight columns in blocks and propagates rounding errors to the remaining columns. Algorithm 1 highlights the changes to the calibration input in blue. The black lines use the upper-triangular Cholesky factor U, so the same factor supplies both the diagonal error normalization and the updates. Here $d_{\text{row}} = C_o$, $d_{\text{col}} = C_i$, and slice endpoints are exclusive.

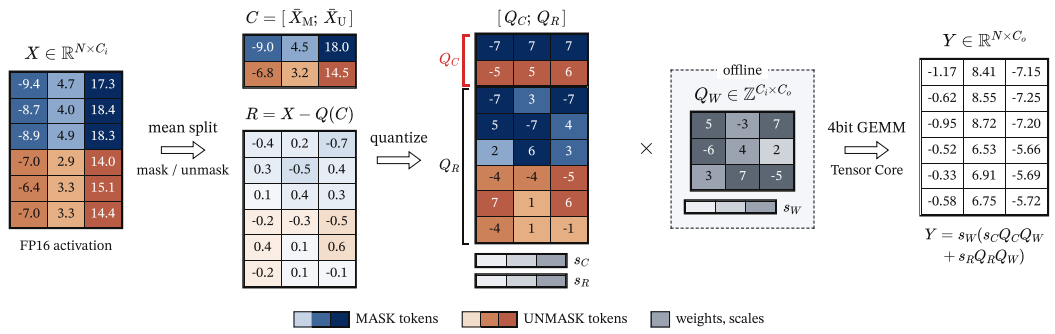


Figure 5: **MUENGINE's fused data path.** Stacking two center rows with the centered activations lets one 4-bit matrix multiplication compute both components.

Difference from certainty-guided calibration. CGQ reweights Hessian contributions by mask status and confidence (Xu et al., 2026), whereas MuCal constructs its Hessian from MuZero's reconstructed $\widetilde{X}_t$. Thus, even when both methods observe the same rollout, CGQ changes token importance while MuCal changes the representation against which rounding error is measured. Whether the two mechanisms compose favorably remains an open empirical question.

Layer-wise calibration pass. For each target linear layer, we replay the stored mask-labelled snapshots, reconstruct $\widetilde{X}_t$, and accumulate its statistics before applying the usual GPTQ block order. Once a layer is rounded, its snapshots and statistics can be discarded. The one-time offline pass therefore produces only packed four-bit weights and scales, while serving recomputes MuZero's state-specific centers for each forward pass. For attention key and value paths, statistics are restricted to masked query positions, which drive the denoising update.

4.3 MUENGINE: ONE FUSED 4-BIT KERNEL

Representing two centers need not require two weight passes. Separate GEMMs would read the same matrix twice, whereas MuEngine stacks center and residual codes into one activation operand, multiplies it by the packed weight once, and separates the resulting rows in the epilogue, as Figure 5 illustrates.

One multiplication for both contributions. The center and residual contributions share the same weights, so stacking their low-bit codes lets one multiplication compute both:

$$Z = \begin{bmatrix} Q_C \\ Q_R \end{bmatrix}, \qquad G = ZQ_W = \begin{bmatrix} G_C \\ G_R \end{bmatrix}. \tag{7}$$

where Q_C and Q_R contain two center rows and N residual rows, Q_W holds the weight codes, and G_C, G_R are their respective outputs. The epilogue restores scales and selects each token's center through $Y_i = \text{dequant}(G_R[i]) + \text{dequant}(G_C[s_t(i)])$, avoiding a second weight pass.

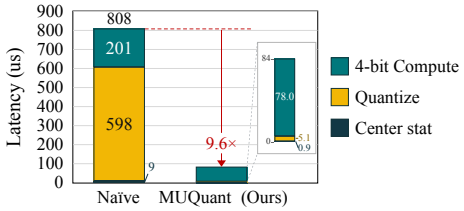


Figure 6: **MuEngine latency breakdown.** Fusing the low-bit path reduces latency from 808 to 84 μs, a 9.6× improvement over the naive implementation.

Cost of preserving two origins. Packing adds two rows to an N-row GEMM, or $2/N$ additional arithmetic, while retaining a single weight read. The center contribution is computed once per state and reused across its tokens. Figure 6 shows why fusion matters in practice: separate center-statistics and quantization operations dominate the naive implementation.

5 EXPERIMENTS

Table 1: **INT4 W4A4 accuracy on LongBench QA.** Blue rows denote MUQuant. Within each model and calibration group, bold and underlined scores indicate the best and second-best results, respectively. Drop is the decrease in macro F1 from the same model's BF16 score, with the smallest decrease in each group highlighted in green.

Model	Method	2wiki	hotpot	musique	narr.	qasper	trivia	macro	**Drop**
Nemotron-Labs-Diffusion-8B	BF16	41.78	39.23	18.58	20.70	39.56	82.85	40.45	–
	RTN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	↓40.45
	QuaRot	24.61	20.20	11.63	11.99	**30.43**	60.60	26.58	↓13.87
	MUQuant (w/o MuCal)	**28.96**	**28.56**	**12.72**	**16.90**	27.71	**66.41**	**30.21**	↓10.24
	DLLMQuant	34.57	29.90	**17.79**	9.99	20.65	69.46	30.39	↓10.06
	QuaRot + GPTQ	31.46	28.16	17.66	15.35	26.15	77.29	32.68	↓7.77
	MUQuant	**36.37**	**33.20**	17.40	**18.13**	**27.59**	**78.14**	**35.14**	↓5.31
SDAR-8B-64	BF16	45.48	43.11	24.90	22.77	17.67	89.20	40.52	–
	RTN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	↓40.52
	QuaRot	19.83	33.71	12.42	13.21	13.98	72.93	27.68	↓12.84
	MUQuant (w/o MuCal)	**33.70**	**33.90**	**20.51**	**20.32**	**15.00**	**78.50**	**33.65**	↓6.87
	DLLMQuant	29.64	37.61	15.74	**19.46**	14.21	**87.51**	34.03	↓6.49
	QuaRot + GPTQ	28.67	34.31	11.17	17.06	**17.78**	86.60	32.60	↓7.92
	MUQuant	**44.32**	**42.74**	**23.93**	15.65	13.33	86.18	**37.69**	↓2.83
DiffusionGemma-26B	BF16	45.52	42.42	22.84	19.73	36.07	89.66	42.71	–
	RTN	0.08	0.00	0.00	0.00	0.00	0.57	0.11	↓42.60
	QuaRot	4.14	2.83	0.00	1.03	5.69	6.00	3.28	↓39.43
	MUQuant (w/o MuCal)	**26.25**	**27.78**	**4.00**	**11.19**	**13.64**	**60.60**	**23.91**	↓18.80
	DLLMQuant	0.00	0.00	0.00	0.00	0.49	0.20	0.12	↓42.59
	QuaRot + GPTQ	20.13	23.51	4.84	6.96	14.68	61.14	21.88	↓20.83
	MUQuant	**37.69**	**34.17**	**19.05**	**17.31**	**23.25**	**80.11**	**35.26**	↓7.45

5.1 EXPERIMENTAL SETUP

Models and tasks. Following LoSA (Xi et al., 2026a), we evaluate Nemotron-Labs-Diffusion-8B (NVIDIA, 2026), SDAR-8B-64 (Cheng et al., 2025), and DiffusionGemma-26B on six LongBench QA tasks (Bai et al., 2024) with 4K-token prefixes and official task scores. Appendix D identifies the task sources, and Appendix C reports the JevBench evaluation. The subset comprises 2WikiMQA, HotpotQA, MuSiQue, NarrativeQA, Qasper, and TriviaQA. We use the LongBench versions of these tasks rather than the full original datasets, so the reported scores characterize the shared 4K-prefix evaluation setting.

Quantization and calibration. INT4 uses symmetric per-token activation quantization and weight groups of 128, while NVFP4 uses its native grid and scaling hierarchy. Following DLLMQuant (Xu et al., 2026), all calibrated methods use the same 128 WinoGrande (Sakaguchi et al., 2020) examples. MuCal collects denoising rollouts from these examples for offline weight rounding. Its calibration statistics use the MuZero-reconstructed activations, while the subsequent weight updates follow GPTQ. INT4 and NVFP4 results are reported separately because their codebooks and scaling schemes differ. Comparisons within each format therefore assess the methods under the corresponding numerical representation, rather than treating all four-bit formats as interchangeable.

Baselines and reporting. RTN, QuaRot, and MUQuant (w/o MuCal) are calibration-free. DLLMQuant, QuaRot+GPTQ, and full MUQuant use calibration. Macro F1 averages the six tasks, and Drop measures its decrease from the same model's BF16 score. Each task contributes equally to the macro average, regardless of its number of examples. The calibration-free rows compare activation representations without offline weight fitting, whereas the calibrated rows evaluate the combined representation and calibration procedure. We retain per-task scores to distinguish broad improvements from gains concentrated in a subset of tasks.

Table 2: **NVFP4 W4A4 accuracy on LongBench QA.** Blue rows denote MUQuant. Within each model and calibration group, bold and underlined scores indicate the best and second-best results, respectively. Drop is the decrease in macro F1 from the same model's BF16 score, with the smallest decrease in each group highlighted in green.

Model	Method	2wiki	hotpot	musique	narr.	qasper	trivia	macro	**Drop**
Nemotron-Labs-Diffusion-8B	BF16	41.78	39.23	18.58	20.70	39.56	82.85	40.45	–
	RTN	0.00	0.00	0.11	0.00	0.00	0.00	0.02	↓40.43
	QuaRot	**39.60**	**36.22**	**17.89**	**18.64**	30.90	66.20	**34.91**	↓5.54
	MUQuant (w/o MuCal)	32.13	27.69	15.57	17.86	**31.53**	**76.07**	33.48	↓6.97
	DLLMQuant	37.78	37.06	26.86	**18.49**	**32.31**	69.47	37.00	↓3.45
	QuaRot + GPTQ	**44.68**	28.55	20.04	15.78	27.92	69.20	34.36	↓6.09
	MUQuant	40.58	**40.09**	**27.07**	16.87	30.89	**82.67**	**39.69**	↓0.76
SDAR-8B-64	BF16	45.48	43.11	24.90	22.77	17.67	89.20	40.52	–
	RTN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	↓40.52
	QuaRot	**37.63**	35.87	16.04	**21.37**	10.00	85.65	34.43	↓6.09
	MUQuant (w/o MuCal)	32.07	**42.41**	**17.69**	18.87	**12.00**	**87.20**	**35.04**	↓5.48
	DLLMQuant	39.60	41.59	20.70	16.11	**18.33**	90.20	37.76	↓2.76
	QuaRot + GPTQ	40.67	36.09	**24.59**	17.19	15.05	83.87	36.24	↓4.28
	MUQuant	**46.12**	**43.75**	20.69	**17.55**	15.33	**90.87**	**39.05**	↓1.47
DiffusionGemma-26B	BF16	45.52	42.42	22.84	19.73	36.07	89.66	42.71	–
	RTN	0.00	0.00	0.00	0.00	0.00	1.11	0.19	↓42.52
	QuaRot	**46.64**	**34.70**	11.98	18.65	23.83	**82.39**	36.37	↓6.34
	MUQuant (w/o MuCal)	44.79	32.90	**16.73**	**19.88**	**26.46**	79.93	**36.78**	↓5.93
	DLLMQuant	0.00	0.00	0.00	0.11	0.93	1.21	0.38	↓42.33
	QuaRot + GPTQ	45.11	32.14	19.64	**22.24**	**34.31**	86.53	40.00	↓2.71
	MUQuant	**46.24**	**37.81**	**19.81**	19.50	31.09	**89.18**	**40.61**	↓2.10

5.2 Accuracy results

Nemotron-Labs-Diffusion-8B. MUQuant improves INT4 accuracy, although calibrated DLLMQuant remains stronger on MuSiQue. Under NVFP4, MUQuant (w/o MuCal) trails QuaRot, whereas full MUQuant approaches BF16, supporting weight calibration against reconstructed activations. The calibrated NVFP4 gain over QuaRot+GPTQ spans five of the six tasks, although 2WikiMQA still favors QuaRot+GPTQ. The higher macro score thus reflects improvements across several tasks without implying uniform dominance.

SDAR-8B-64. Both formats benefit, particularly INT4. Against QuaRot+GPTQ, calibrated INT4 gains on 2WikiMQA, HotpotQA, and MuSiQue but loses on the other tasks, so the improvement is concentrated in multi-hop QA. NVFP4 gains are smaller. Without calibration, the INT4 comparison is more uniform: MUQuant (w/o MuCal) exceeds QuaRot on all six tasks. Calibration changes this task-level balance even as it raises the overall macro score.

DiffusionGemma-26B. MUQuant (w/o MuCal) recovers much of QuaRot's INT4 accuracy loss, while full MUQuant improves every task over QuaRot+GPTQ, though a substantial gap to BF16 remains. NVFP4 gains are more modest. The calibrated variant approaches BF16, but NarrativeQA and Qasper still favor QuaRot+GPTQ. RTN and DLLMQuant obtain near-zero NVFP4 scores. The gap between the INT4 and NVFP4 results also shows that this backbone's sensitivity cannot be characterized by bit width alone.

5.3 Memory footprint and serving latency

On one RTX 5090, MUQuant runs DiffusionGemma-26B $3.2\times$ faster per decode step than NF4 W4A16, with modestly higher memory use (Figure 1). Both quantized models remain GPU-resident. The larger gain over BF16 also includes avoided offloading transfers and cannot be attributed to four-bit arithmetic alone.

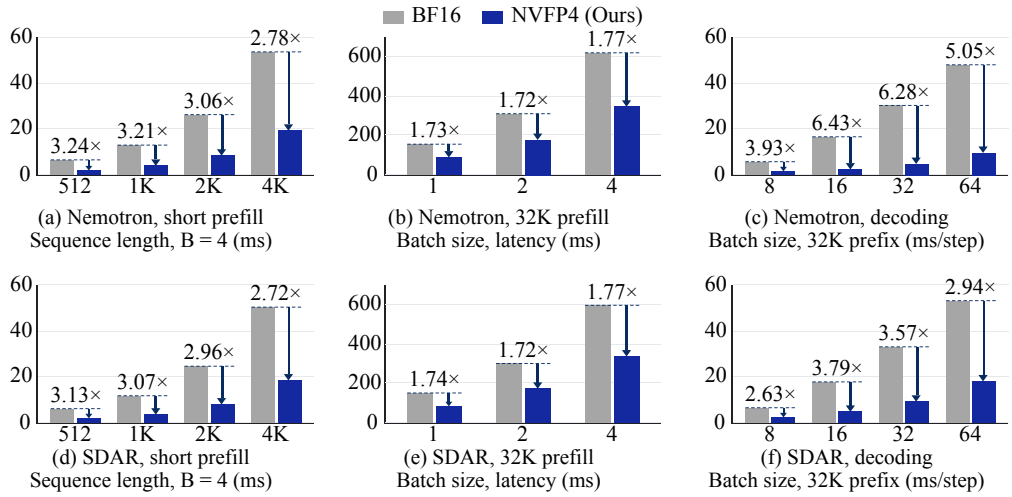


Figure 7: **End-to-end W4A4 efficiency.** Nemotron-Labs-Diffusion-8B (top) and SDAR-8B-64 (bottom). Timings include center statistics, quantization, and decoder work.

5.4 Efficiency results

Figure 7 shows roughly threefold prefill speedups at shorter sequences, with smaller gains at 32K. Decoding benefits across all evaluated batches, most at batch 16. These comparisons include surrounding decoder work and use BF16 as the baseline, unlike the operator-fusion comparison in Figure 6.

5.5 Ablation on center grouping

We compare center groupings on Nemotron-Labs-Diffusion-8B with NVFP4 and no calibration. *2-Random* computes centers for two random token groups, while *Pooled* forms centers from a 50/50 mixture of masked and unmasked tokens. MUQuant instead groups tokens by state, testing whether the grouping matters beyond simply using two centers (Table 3).

Table 3: **Center-grouping ablation.** Nemotron-Labs-Diffusion-8B, NVFP4 without calibration. Drop is the macro-F1 point decrease from BF16 (40.45).

Grouping	2wiki	hotpot	musique	narr.	qasper	trivia	macro	**Drop**
2-Random	27.35	25.73	9.45	16.53	21.70	67.79	28.09	↓12.36
Pooled	29.00	24.46	9.79	15.47	22.49	65.92	27.85	↓12.60
MUQuant (w/o MuCal)	**32.13**	**27.69**	**15.57**	**17.86**	**31.53**	**76.07**	**33.48**	↓6.97

State-specific grouping leads on every task, whereas random grouping and balanced mixing perform similarly. The benefit therefore depends on preserving the distinct populations in Figure 3, rather than on multiple centers alone. Omitting MuCal isolates this comparison from weight calibration.

6 Conclusion

MUQuant uses the masked or unmasked state of each token to select an origin for four-bit activation quantization. MuCal calibrates weights against this operator on denoising rollouts, and MuEngine evaluates its center and residual terms in one packed multiplication. The resulting method improves INT4 accuracy across three backbones and reduces latency in the evaluated execution settings.

AI USE STATEMENT

We used generative AI tools to assist with grammar and language editing. The authors verified all AI-assisted content and take full responsibility for the final manuscript.

REFERENCES

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 (ICLR)*, 2025. URL https://arxiv.org/abs/2503.09573.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. QuaRot: Outlier-free 4-bit inference in rotated LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL https://arxiv.org/abs/2107.03006.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Benchmark Heaven. JevBench: A benchmark for typed decision models. Benchmark repository and scoring documentation, 2026. URL https://github.com/fstandhartinger/jevbench. Accessed September 24, 2026.

Shuang Cheng, Yihan Bian, Dawei Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, and Bowen Zhou. SDAR: A synergistic diffusion–autoregression paradigm for scalable sequence generation. *arXiv preprint arXiv:2510.06303*, 2025.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2307.08691.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2205.14135.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2021. URL https://aclanthology.org/2021.naacl-main.365/.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2208.07339.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. SpQR: A sparse-quantized representation for near-lossless LLM weight compression. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2306.03078.

Elias Frantar, Sidak Pal Singh, and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2208.11580.

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

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *International Conference on Computational Linguistics (COLING)*, 2020. URL https://aclanthology.org/2020.coling-main.580/.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017. URL https://aclanthology.org/P17-1147/.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018. URL https://aclanthology.org/Q18-1023/.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2023. URL https://arxiv.org/abs/2309.06180.

Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. SVDQuant: Absorbing outliers by low-rank components for 4-bit diffusion models. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2411.05007.

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. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024. URL https://arxiv.org/abs/2306.00978.

Lian Liu, Haimeng Ren, Long Cheng, Zhaohui Xu, Yudong Pan, Mengdi Wang, Xiaowei Li, Yinhe Han, and Ying Wang. COMET: Towards practical W4A4KV4 LLM serving. *arXiv preprint arXiv:2410.12168*, 2024.

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 (ICLR)*, 2025.

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyan Wei, Shaobo Wang, Yichen Zhu, and Linfeng Zhang. dLLM-Cache: Accelerating diffusion large language models with adaptive caching. In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2506.06295.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *International Conference on Machine Learning (ICML)*, 2024. URL https://proceedings.mlr.press/v235/lou24a.html.

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. *arXiv preprint arXiv:2502.09992*, 2025.

NVIDIA. Nemotron-Labs-Diffusion: A tri-mode language model. Technical report, NVIDIA, 2026. URL https://huggingface.co/nvidia/Nemotron-Labs-Diffusion-8B.

OpenJev contributors. OpenJev: Open, Jev-compatible System One decision server on DiffusionGemma. Software repository, 2026. URL https://github.com/razorback16/openjev. Accessed September 24, 2026.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2406.07524.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *AAAI Conference on Artificial Intelligence*, 2020. URL https://doi.org/10.1609/aaai.v34i05.6399.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. OmniQuant: Omnidirectionally calibrated quantization for large language models. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2308.13137.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. URL https://aclanthology.org/2022.tacl-1.31/.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. QuIP#: Even better LLM quantization with hadamard incoherence and lattice codebooks. In *International Conference on Machine Learning (ICML)*, 2024. URL https://proceedings.mlr.press/v235/tseng24a.html.

TypeSafe AI. Jev: A System One decision model. Product documentation, 2026. URL https://typesafe.ai/. Accessed September 24, 2026.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009. URL https://doi.org/10.1145/1498765.1498785.

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 (ICLR)*, 2026. URL https://arxiv.org/abs/2505.22618.

Haocheng Xi, Harman Singh, Yuezhou Hu, Coleman Hooper, Rishabh Tiwari, Aditya Tomar, Minjae Lee, Wonjun Kang, Michael W. Mahoney, Chenfeng Xu, Kurt Keutzer, and Amir Gholami. LoSA: Locality aware sparse attention for block-wise diffusion language models. In *International Conference on Machine Learning (ICML)*, 2026a. URL https://arxiv.org/abs/2604.12056.

Haocheng Xi, Shuo Yang, Yilong Zhao, Muyang Li, Han Cai, Xingyang Li, Yujun Lin, Zhuoyang Zhang, Jintao Zhang, Xiuyu Li, Zhiying Xu, Jun Wu, Chenfeng Xu, Ion Stoica, Song Han, and Kurt Keutzer. Quant VideoGen: Auto-regressive long video generation via 2-bit KV-cache quantization. In *International Conference on Machine Learning (ICML)*, 2026b. URL https://arxiv.org/abs/2602.02958.

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 (ICML)*, 2023.

Chen Xu, Zhixuan Chen, and Dawei Yang. DLLMQuant: A post-training quantization framework tailored for diffusion-based large language models. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, volume 306 of *Proceedings of Machine Learning Research*, 2026. URL https://icml.cc/virtual/2026/poster/62533.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018. URL https://aclanthology.org/D18-1259/.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2206.01861.

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*, 2025a.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025b. URL https://arxiv.org/abs/2501.01005.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate LLM serving. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 6, pp. 196–209, 2024.

A COORDINATE-WISE OBQ UPDATE

OBQ, introduced within Optimal Brain Compression (Frantar et al., 2022), quantizes one weight coordinate at a time and uses the inverse Hessian to compensate its rounding error in the remaining coordinates. For coordinate q, the induced error and optimal correction are

$$\epsilon_q = \frac{(w_q - \mathrm{quant}(w_q))^2}{[H^{-1}]_{qq}}, \qquad \delta w = -\frac{w_q - \mathrm{quant}(w_q)}{[H^{-1}]_{qq}} H^{-1}_{:,q}. \tag{8}$$

GPTQ applies these updates in blocks and uses a Cholesky reformulation to make the procedure practical and numerically stable at transformer scale (Frantar et al., 2023).

B PROGRESSIVE QUANTIZATION AND STATE-CONDITIONED CENTERING

Per-token scaling and center subtraction address different parts of the quantization problem. A scale determines the spacing of representable values, whereas a center determines which component of the activation must be represented by those values. MuZero retains per-token scales for the residuals. Its purpose is to make those residuals easier to quantize by first representing a channel-wise component shared by tokens in the same state. This section develops that interpretation through progressive residual quantization and explains why the shared component can be evaluated with only two additional rows in the linear multiplication. We suppress the decoding-step subscript throughout.

B.1 WHY A PER-TOKEN SCALE DOES NOT REPLACE A CENTER

Giving each token its own scale prevents a large activation in one token from setting the resolution of every other token. It does not, however, prevent a large channel within that token from setting the resolution of its remaining channels. For the symmetric integer quantizer used in the main text, the scale and the rounding-error bound are

$$s_i = \frac{\|x_i\|_\infty}{q_{\max}}, \qquad \|Q_b(x_i) - x_i\|_\infty \le \frac{s_i}{2}, \qquad q_{\max} = 2^{b-1} - 1. \tag{9}$$

where $x_i \in \mathbb{R}^{C_i}$ is one token's channel vector. The bound assumes round-to-nearest with an exact scale and no clipping beyond the range set by the maximum. If a large component of x_i is shared across tokens, repeatedly encoding that component uses range that could otherwise resolve their smaller differences.

MuZero represents the shared component separately before selecting the residual scale. The relevant comparison is therefore

$$s_i^{\mathrm{direct}} = \frac{\|x_i\|_\infty}{q_{\max}}, \qquad s_i^{\mathrm{res}} = \frac{\|x_i - Q_b(c_{s(i)})\|_\infty}{q_{\max}}. \tag{10}$$

where $c_{s(i)}$ is the center selected by token i's mask state. Both scales remain token-specific. The second can be smaller because it is fitted to the variation around a shared channel vector rather than to the original activation. This center is not a scalar zero point: its coordinates can differ across channels, so neither one multiplicative scale nor one scalar shift per token generally expresses the same transformation.

The distinction also separates additive centering from channel-wise rescaling. A shared diagonal transform changes the magnitude of each channel for all tokens, but cannot subtract different offsets from masked and unmasked tokens in that channel. Here, smoothing means removing a representable shared component before quantizing the remaining variation.

B.2 THE PROGRESSIVE RESIDUAL-QUANTIZATION VIEW

One way to improve a coarse quantization is to encode what it misses in a second stage. Repeating this construction gives a general residual-quantization expansion:

$$\begin{aligned} R^{(0)} &= X, & A^{(k)} &= Q_k(R^{(k-1)}), \\ R^{(k)} &= R^{(k-1)} - A^{(k)}, & \widetilde{X}^{(n)} &= \sum_{k=1}^{n} A^{(k)} = X - R^{(n)}. \end{aligned} \tag{11}$$

where k indexes refinement stages, Q_k returns a reconstructed approximation, and $R^{(k)}$ is what remains after stage k. The familiar first residual $R^{(1)} = X - Q_1(X)$ is the first step of this expansion. Later stages devote their representational capacity to the remaining error instead of encoding the original range again. This identity describes the decomposition, not an unconditional guarantee that every quantizer or additional stage improves downstream accuracy.

Quant VideoGen applies a related coarse-to-fine construction to video KV-cache compression (Xi et al., 2026b). Its Semantic-Aware Smoothing groups similar tokens with k-means and subtracts their centroids. Progressive Residual Quantization repeats that grouping and subtraction on successive residuals, then quantizes the final residual while retaining the stage-wise centroids and assignments for reconstruction. Thus, its stages use centroid-based approximations rather than simply applying the same scalar quantizer repeatedly to the full tensor. This provides a precedent for interpreting center subtraction as a coarse representation followed by residual coding.

For W4A4 linear layers, the difficulty is evaluating the resulting expansion efficiently. If every stage contributes a distinct N-row activation operand, evaluating all terms against the weight matrix requires nN rows of multiplication. Packing can combine these rows into one GEMM, but it does not remove their arithmetic cost. MuZero restricts the coarse component to two shared centers so that its contribution need not be evaluated independently for every token.

B.3 MuZero as a shared coarse stage

The mask indicator already supplies the grouping, so the first stage need not discover assignments by clustering. We estimate one channel vector for each nonempty state and quantize those vectors once. Broadcasting them to their tokens gives the coarse approximation, followed by a token-wise residual stage:

$$M = \begin{bmatrix} c_{\mathrm{M}} \\ c_{\mathrm{U}} \end{bmatrix}, \quad \overline{M} = Q_4(M), \quad C = SM, \quad R = X - S\overline{M}, \quad \widetilde{X} = S\overline{M} + Q_4(R). \tag{12}$$

where $M \in \mathbb{R}^{2\times C_i}$ contains the state centers and $S \in \{0,1\}^{N\times 2}$ selects the appropriate state for each token. Each row of S has one nonzero entry. The implementation uses the existing mask indicator rather than materializing this matrix. If only one state is present, only its center is needed. By construction, the broadcast reconstruction $S\overline{M}$ is the $Q_4(C)$ term in Equation 5.

The first approximation is therefore $Q_4(C)$, not a separately quantized copy of every row of X. The residual $R = X - Q_4(C)$ retains the information that this shared approximation cannot represent, and each residual row receives its own quantization scale. MuZero can thus be viewed as a structured two-component residual representation, rather than a full multi-stage quantizer applied independently to all tokens.

Subtracting the reconstructed center is essential when the center itself is low-bit. If the residual were formed around the unquantized center, its reconstruction error would contain both center rounding and residual rounding. Using the center actually consumed by the multiplication instead yields

$$\widetilde{X} - X = Q_4(R) - R, \qquad R = X - Q_4(C). \tag{13}$$

where the equality is in real arithmetic before implementation-level accumulation error. Center rounding has not disappeared from the problem. It is included in the residual to be quantized, rather than becoming a separate uncompensated error term.

B.4 When centering improves quantization resolution

The benefit depends on whether the state center captures a substantial component of the token activation. Writing a token as $x_i = c_{s(i)} + \delta_i$ makes this dependence explicit:

$$r_i = \delta_i + e_{s(i)}, \qquad e_a = c_a - Q_4(c_a), \qquad \|r_i\|_\infty \le \|\delta_i\|_\infty + \|e_{s(i)}\|_\infty. \tag{14}$$

where δ_i is the token-specific deviation and e_a is center-rounding error for state a. When their combined magnitude is small relative to the original activation, Equation 10 gives a finer residual grid even though the original quantizer already used per-token scales.

For example, a sufficient condition for the residual range to be smaller is

$$\|c_{s(i)}\|_\infty > 2\|\delta_i\|_\infty + \|e_{s(i)}\|_\infty. \tag{15}$$

The reverse triangle inequality gives $\|x_i\|_\infty \geq \|c_{s(i)}\|_\infty - \|\delta_i\|_\infty$, which then exceeds the upper bound on $\|r_i\|_\infty$. This is a sufficient condition for a smaller rounding-error bound, not a claim that every token satisfies it or that every actual rounding error decreases. A group mean minimizes squared residual energy for fixed assignments, but does not necessarily minimize the largest residual coordinate. The grouping ablation in Table 3 tests the practical choice of state-specific centers rather than assuming that any partition will be effective.

The uniform-grid bounds above describe INT4. NVFP4 has a nonuniform codebook and additional scale quantization, so the same bound should not be transferred to it unchanged. The coarse-plus-residual decomposition and the cancellation in Equation 13 still hold for its reconstructed values. Whether that representation improves accuracy depends on the resulting residual distribution and the quantizer used.

B.5 Why the coarse stage adds only two GEMM rows

Because all tokens in one state share their coarse vector, its linear projection can be computed once and selected for each token at the output. Distributing the linear operator over the two components gives

$$\widetilde{X}\overline{W}^\top = S\big(\overline{MW}^\top\big) + Q_4(R)\overline{W}^\top, \qquad \overline{W} = Q_4(W). \tag{16}$$

where $\overline{MW}^\top$ has only two rows, while the residual projection has N. MuEngine implements these terms with packed low-bit codes and restores their scales in the epilogue, as described in Section 4.3.

With $K \leq 2$ nonempty states, the ideal dense multiplication count increases from $2NC_iC_o$ to $2(N + K)C_iC_o$, a relative arithmetic increase of K/N. Thus, "two extra tokens" means two additional activation rows in the GEMM, not two additional sequence positions or only two tokens' worth of total preprocessing. Center estimation and residual formation still touch the activation tensor, and output selection, scaling, and reconstruction still require work. Tile padding and memory traffic can also affect realized latency. Fusion is what makes these operations practical without a second full weight pass.

From this perspective, the efficiency of MuZero comes from sharing the coarse stage. Per-token scales preserve local resolution in the residual, while state-specific centers carry a common component through a projection whose row count is independent of sequence length. The two mechanisms are complementary rather than redundant.

C TYPED-DECISION EVALUATION ON JEVBENCH

Jev (TypeSafe AI, 2026) is a decision model that returns typed answers and their probabilities to structured questions instead of generating free-form text. To evaluate MUQuant in this setting, we use the 231-item public set of JevBench (Benchmark Heaven, 2026), a benchmark for such decisions. We evaluate DiffusionGemma-26B using INT4 W4A4 kernels and an OpenJev-style single read-only denoising step (OpenJev contributors, 2026), reading label probabilities at one masked answer slot in an otherwise unmasked 64-position canvas. Table 4 reports Intelligence and hard-tier accuracy with BF16 as the reference.

Table 4: **JevBench public-set results.** DiffusionGemma-26B, 231 items, INT4 W4A4. Higher is better for both metrics. Bold and underlined scores indicate the best and second-best quantized results, respectively.

Method	Intelligence	Hard accuracy (%)
BF16	84.4	64.0
RTN (no rotation)	36.7	38.7
QuaRot	70.4	46.8
MUQuant (w/o MuCal)	<u>77.4</u>	<u>49.5</u>
DLLMQuant	32.9	36.9
QuaRot + GPTQ	76.8	<u>49.5</u>
MUQuant	**79.4**	**55.9**

Figures 8–11 show two selected hard examples where BF16 and MUQuant answer correctly but QuaRot+GPTQ does not. Input and output panels are enlarged separately, with all supplied content retained.

(a) Input to the model

SYSTEM

Answer a fixed set of questions about the state the user provides. Each question lists its allowed answers; reply with exactly one label.

Question q1: Estimate the outcome of the appointment.

A: attended (The patient attends the appointment.)

B: late_cancellation (The patient cancels < 24 h before.)

C: no_show (The patient does not attend and does not cancel.)

Reply with one line per question, formatted as "id: label".

USER

Appointment outcome statistics (1,000 appointments):

Reminder status at 24 h before	Appts	Attended	No-show	Late cancel
Confirmed by patient (SMS "Y")	200	160	16	24
Reminder delivered, no reply	500	340	110	50
Reminder not delivered	300	180	84	36
All	1,000	680	210	110

Rule: use the row matching the reminder status at the 24-hour mark; later changes are ignored.

Appointment: Mon 2026-09-21 09:40 (24-hour mark: Sun 09:40).

SMS delivered Sat 09:41; patient reply "Y" received Sun 08:15.

Receptionist: "With the overall 21 % no-show rate I'd double-book that slot."

DECODER INPUT (CANVAS)

<channel> | thought | \n | <channel> | q | 1 | : | **[MASK]** | <turn> | <pad> | …

[MASK] marks the answer slot (random noise token per seed).

Figure 8: **JevBench example 1: input.** Probability estimation with a timing trap (`hard-opus-b-probability-04`). The original prompt and decoder canvas are shown.

(b) Label probabilities at the slot, averaged over 4 canvas seeds

	BF16 full precision	**QuaRot + GPTQ** INT4 W4A4	**MUQuant + MuCal** INT4 W4A4
attended†	**0.996**	0.401	**0.956**
late_cancellation	0.004	**0.518**	0.036
no_show	0.000	0.080	0.008
Prediction	attended ✓	late_cancellation ✗	attended ✓

(c) Top-5 tokens at the slot (seed 0)

#	**BF16**		**QuaRot + GPTQ**		**MUQuant + MuCal**	
1	'A'	0.993	'B'	0.501	'A'	0.946
2	'B'	0.002	'A'	0.370	'B'	0.448
3	':'	0.001	'C'	0.082	'C'	0.004
4	'<turn>'	0.001	','	0.008	':'	0.001
5	'<eos>'	0.001	':'	0.007	'<turn>'	0.001

Figure 9: **JevBench example 1: output.** Label probabilities are averaged over four canvas seeds, with one denoising step per read. Top-5 tokens are shown for seed 0.

(a) Input to the model

SYSTEM

Question q1: Does the response fully and correctly satisfy every explicit requirement?

A: yes

B: no

Reply with one line per question, formatted as "id: label".

USER

Request:

Subtotal $19.95. Apply a 15 % discount, then 8.25 % tax to the discounted amount. Do not round the intermediate amount. Round the final charge to cents using half-up, and show the derivation.

Response:

Discounted = 19.95 × 0.85 = 16.9575.

Taxed = 16.9575 × 1.0825 = 18.35649375.

Rounded half-up to cents, the final charge is $18.35.

DECODER INPUT (CANVAS)

<channel>	thought	\n	<channel>	q	1	:	**[MASK]**	<turn>	<pad>	…

[MASK] marks the answer slot (random noise token per seed).

Figure 10: **JevBench example 2: input.** Judging a calculation with a rounding error (`hard-sol-c-judge_hard-13`). The original prompt and decoder canvas are shown.

(b) Label probabilities at the slot, averaged over 4 canvas seeds

	BF16 full precision	**QuaRot + GPTQ** INT4 W4A4	**MUQuant + MuCal** INT4 W4A4
no†	**0.973**	0.486	**0.990**
yes	0.027	**0.514**	0.010
Prediction	no ✓	yes ✗	no ✓

(c) Top-5 tokens at the slot (seed 0)

#	**BF16**		**QuaRot + GPTQ**		**MUQuant + MuCal**	
1	'B'	0.970	'A'	0.506	'B'	0.986
2	'A'	0.024	'B'	0.472	'A'	0.010
3	':'	0.002	':'	0.008	':'	0.001
4	'<turn>'	0.001	' yes'	0.006	'<turn>'	0.001
5	' yes'	0.001	'<turn>'	0.003	' yes'	0.001

Figure 11: **JevBench example 2: output.** Label probabilities are averaged over four canvas seeds, with one denoising step per read. Top-5 tokens are shown for seed 0.

D ADDITIONAL RELATED WORK AND TASK REFERENCES

Discrete diffusion. D3PM studies structured corruption in discrete state spaces (Austin et al., 2021). Masked diffusion language models simplify the training objective (Sahoo et al., 2024), while SEDD learns ratios of the discrete data distribution (Lou et al., 2024). These works concern the generative model and its training, whereas MUQuant quantizes a trained model without changing its denoising procedure.

Quantization and execution. AWQ protects activation-sensitive weight channels through scaling (Lin et al., 2024), and SpQR preserves outlier weights in higher precision (Dettmers et al., 2024). OmniQuant learns clipping and equivalent transformations (Shao et al., 2024), while QuIP# combines Hadamard incoherence processing with lattice codebooks (Tseng et al., 2024). These approaches provide context for low-bit representation design rather than additional experimental baselines here. Efficient execution also depends on the surrounding attention implementation. FlashAttention and FlashAttention-2 reduce attention's memory-access costs and improve GPU work partitioning (Dao et al., 2022; Dao, 2024). PagedAttention addresses KV-cache memory management (Kwon et al., 2023), and FlashInfer provides customizable inference attention kernels (Ye et al., 2025b). These optimizations target different components from MuEngine's linear operator.

LongBench task sources. The six-task subset used in Tables 1 and 2 comprises 2WikiMQA (Ho et al., 2020), HotpotQA (Yang et al., 2018), MuSiQue (Trivedi et al., 2022), NarrativeQA (Kočiský et al., 2018), Qasper (Dasigi et al., 2021), and TriviaQA (Joshi et al., 2017). We use their LongBench versions (Bai et al., 2024), rather than reporting results on the full original datasets. The 4K-prefix protocol and calibrated baselines are described in Section 5.