QuMix: Query-Guided Mixed-Precision KV-Cache Quantization for Block Diffusion Language Models

Anonymous authors
Paper under double-blind review

Abstract

Block diffusion lets diffusion language models (dLLMs) reuse completed blocks through a key–value (KV) cache, but storing and repeatedly reading this growing cache limits context length, batch size, and decoding speed. KV-cache quantization reduces these costs, yet has been studied almost entirely in autoregressive (AR) models. Applying an existing AR quantizer to a block-diffusion model, we find that two-bit quantization preserves nearly all of its original TriviaQA score, whereas reducing the precision to one bit causes a large drop. This motivates mixed precision, which lowers average storage by retaining more bits for sensitive components and using fewer bits elsewhere. Such allocation must preserve information that attention relies on, including the key of the attention sink—a token that many heads repeatedly attend to. AR models typically place this sink at the first token, making it possible to protect by position, but sink positions can vary in dLLMs. We therefore introduce QuMix, which uses queries to guide precision allocation without assuming a sink position. From calibration queries and keys, QuMix constructs a query-weighted basis and assigns more bits to directions whose errors have a larger effect on attention scores, without model training. To our knowledge, this is the first study of KV-cache quantization for block diffusion. QuMix retains 98–99% of the 16-bit LongBench score across three models at an average of 1.99 bits per cache element. A custom CUDA kernel that attends directly to the compressed cache supports a $10.5\times$ larger maximum batch and $2.3\times$ faster denoising steps at the same batch size (batch 15) than BF16 FlashAttention4 on an RTX PRO 6000 GPU.

1 Introduction

Diffusion language models (dLLMs) are becoming competitive with autoregressive (AR) models on language understanding, reasoning, and coding tasks (Nie et al., 2025; Ye et al., 2025). Block diffusion combines the two approaches: it generates blocks sequentially and refines the tokens within each block together through diffusion (Arriola et al., 2025; Cheng et al., 2026). Because completed blocks no longer change, their keys and values can be stored in a key–value (KV) cache and reused by later blocks instead of being recomputed. Longer contexts, however, make this cache increasingly expensive. Its size grows with both context length and batch size, limiting how much context and how many requests fit on a GPU (Figure 1a). Each denoising step also reads the cache to compute attention for the active block (Xi et al., 2026), so repeatedly accessing it can limit decoding speed. Reducing these storage and access costs is therefore important for efficient dLLM inference.

KV-cache quantization reduces these costs by storing and transferring fewer bits per cached element, but has been studied almost entirely in AR models. To examine whether existing AR quantizers can provide these savings in block diffusion, we apply KIVI (Liu et al., 2024b), a method that quantizes keys and values to low precision, to the block-diffusion model SDAR. With two-bit quantization, SDAR retains nearly all of its original TriviaQA score (Figure 1b). This result motivates seeking further savings, but lowering KIVI's precision to one bit causes a large score drop. Mixed precision offers one way to pursue these savings: components that tolerate larger errors can use fewer bits, allowing sensitive components to retain higher precision within a smaller storage budget.

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.

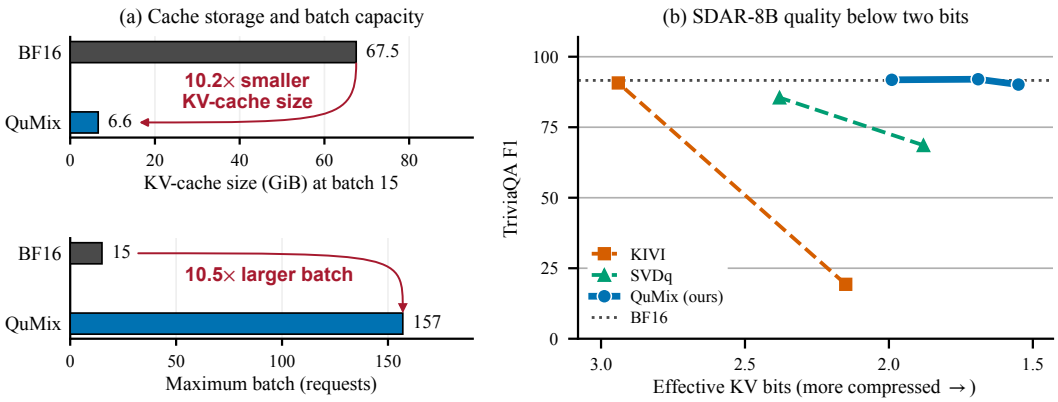


Figure 1: **a)** QuMix reduces KV-cache size and increases batch capacity. **b)** QuMix preserves SDAR's TriviaQA score below two bits per cache element.

The challenge is determining which components need those extra bits. For example, SVDq (Hong et al., 2025), an existing mixed-precision quantizer, allocates precision according to key variance, but SDAR still loses more than a quarter of its BF16 TriviaQA score at its sub-two-bit setting. This loss shows that allocating bits selectively does not by itself ensure accurate block-diffusion inference; the allocation must protect keys that attention relies on.

Our analysis shows that restoring a small group of early keys recovers most of this loss in SDAR (Section 3.1). These keys include the attention sink, a token that many heads repeatedly attend to. This recovery motivates examining whether positional protection can preserve the keys that block diffusion relies on. In AR models, the sink typically forms at the first token, allowing quantizers to protect it by keeping that token at full precision (Liu et al., 2024a). In dLLMs, however, sink positions can shift during denoising (Rulli et al., 2026), so protecting the first token alone can leave the sink exposed to quantization error (Section 3.2). The challenge is therefore to preserve the key information that attention relies on without assuming where it occurs in the cache.

Queries provide a way to identify this information because they determine how errors in the cached keys affect attention scores (Section 3.3). Building on this criterion, we propose QuMix, a *Qu*ery-guided *Mix*ed-precision KV-cache quantization method for block diffusion. QuMix uses calibration queries and keys to construct a query-weighted basis and assigns more bits to directions whose errors have a larger effect on attention scores. The basis and bit allocation remain fixed across requests, requiring neither model training nor a model-specific rule for locating the sink. To translate the compressed cache into larger batches and faster decoding, we also develop a custom CUDA kernel that computes attention directly over it.

QuMix preserves SDAR's TriviaQA score below two bits (Figure 1b) and retains 98–99% of the 16-bit LongBench score across SDAR, Nemotron, and TraDo at 1.99 bits per cache element. With SDAR-8B at the 1.55-bit setting, the maximum batch size for 32K-token inputs increases from 15 to 157 on a single RTX PRO 6000 GPU (Figure 1a). At batch 15, the largest batch that fits with BF16, a denoising step runs 2.3× faster than with BF16 FlashAttention4 (Zadouri et al., 2026). To our knowledge, this is the first study of KV-cache quantization for block diffusion.

2 Background

2.1 Caching in block diffusion

Block diffusion generates a sequence one block at a time and denoises the tokens within each block together (Arriola et al., 2025; Cheng et al., 2026; Wang et al., 2026). Attention is causal across blocks, so completed blocks never attend to later ones and their keys and values remain fixed throughout the rest of generation. These keys and values can therefore be cached and reused exactly. At each denoising step, the model recomputes only the active block and attends to the cache for context.

Although caching avoids recomputing completed blocks, storing and repeatedly reading their keys and values remains costly as the context grows. Sparse attention reduces how many cache entries are read (Song et al., 2025; Xi et al., 2026), while quantization reduces the number of bits used to store each entry. We study quantization of the completed-block cache and keep the active block, whose states change at every denoising step, at full precision.

2.2 KV-CACHE QUANTIZATION

KV-cache quantization has primarily been developed for AR models, where it reduces memory use by storing keys and values at lower precision. The resulting errors affect attention in different ways: key errors change how attention is distributed across cached tokens, while value errors change the information returned by attention. Existing methods account for the different distributions of keys and values. KIVI quantizes keys per channel across tokens and values per token to accommodate their different outlier patterns (Liu et al., 2024b). To preserve the channel structure of the keys during quantization, KVQuant quantizes them before applying rotary position embeddings (RoPE) (Hooper et al., 2024).

Rather than preserving the original channel structure, rotation-based methods change it to make quantization more accurate. TurboQuant applies a random rotation before quantizing each coordinate, using quantization levels matched to the resulting distribution (Zandieh et al., 2026). These approaches reduce quantization error without changing the model weights. For dLLMs, prior accelerator work has explored four-bit KV-cache quantization in models such as LLaDA (Lou et al., 2026). That setting differs from the completed-block cache studied here, which supports exact reuse under block-causal attention.

2.3 MIXED PRECISION FOR ULTRA-LOW-BIT QUANTIZATION

Reducing precision uniformly gives every cache component the same number of bits, even though some components are more sensitive to error than others. Mixed precision uses this difference to reduce average storage: it retains more bits for sensitive components and compresses the others more aggressively. Its effectiveness depends on the information used to decide which components need those extra bits. SVDq uses key variance, assigning more bits to directions whose values vary widely across cached tokens and dropping directions with little variation at its most compressed settings (Hong et al., 2025). Other methods incorporate query statistics or measured quantization error. MixKVQ combines query relevance with the quantization difficulty of each key channel (Zhang et al., 2026b), while Block-GTQ uses query and key energy for each pair of channels that RoPE rotates together (Liang et al., 2026). RateQuant allocates bits across heads using distortion curves fitted on calibration data (Zuo et al., 2026).

The units receiving those bits can also be changed by transforming the keys before quantization. This allows precision to be assigned to directions that combine information from multiple original channels. Query statistics can guide this transformation as well as the allocation of bits: OSCAR and NOVA-KV use query-informed transforms for quantization at about two bits per element (Zhou et al., 2026; Fernández-Menduiña et al., 2026). SPECTRA instead targets storage below two bits by allocating precision across directions derived from the key projection and its input activations, without query statistics (Zhang et al., 2026a). These methods were developed for AR models; Section 3 examines what must be preserved when applying ultra-low-bit compression to block diffusion.

3 TOWARD ULTRA-LOW-BIT KV-CACHE QUANTIZATION FOR DLLMS

Accurate sub-two-bit KV-cache quantization requires deciding which information can tolerate error and which must be preserved. We investigate this choice in SDAR by testing a key-variance criterion, examining whether important keys can be protected by token position, and using these findings to motivate a query-based allocation criterion.

3.1 IDENTIFYING KEY INFORMATION THAT NEEDS PROTECTION

We first examine which cached keys need higher precision to preserve a dLLM's accuracy under aggressive compression. For this analysis, we use SVDq (Hong et al., 2025), which assigns more

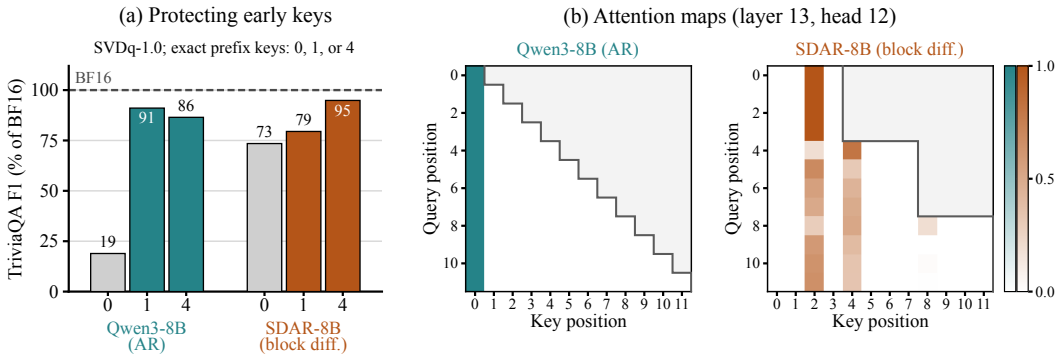


Figure 2: First-token protection recovers most of Qwen3's lost score but little of SDAR's. **a)** Restoring the first four keys, covering SDAR's first block, recovers most of its loss under SVDq-1.0; values remain at two bits. **b)** Attention maps at the same layer and head index; Appendix B.3 reports head selection and first-token attention averaged over all heads.

bits to key directions whose values vary widely across cached tokens and drops directions with little variation at its most compressed setting. At the evaluated SVDq-1.0 setting, which uses about 1.88 bits per cache element, SDAR retains 73% of its BF16 score on LongBench TriviaQA (Bai et al., 2024). Thus, substantial accuracy loss remains despite selective precision allocation.

One potential source of this loss is damage to the attention sink, a token that many heads repeatedly attend to (Xiao et al., 2024; Gu et al., 2025). Errors in its key may therefore disrupt attention even when the affected directions contribute little to key variance. In our SDAR setting, the first four cached keys include the sink's key. To test whether errors in these early keys contribute to the accuracy loss, we restore them to their original full-precision values while leaving all other keys and values unchanged. Restoring this small part of the cache recovers most of the lost score (Figure 2a), showing that preserving these early keys can substantially improve dLLM quantization without increasing precision throughout the cache.

3.2 First-token protection can miss important keys

The next question is how to identify the keys that need this protection. Initial-token preservation in AR models (Liu et al., 2024a) suggests testing a simple rule: keep the first key at full precision. We compare this rule in SDAR and Qwen3 (Yang et al., 2025), an AR model with the same base, using the same LongBench TriviaQA inputs and SVDq setting. Restoring only the first key recovers most of Qwen3's lost score but little of SDAR's (Figure 2a).

The attention maps show why the first-token rule misses SDAR's sink in this setting: it occurs at the third token rather than the first (Figure 2b). Under causal attention, the first token attends only to itself; under SDAR's block-causal attention, it also attends to the rest of its block. The maps and restoration results show that protecting the first token cannot be assumed to preserve important key information in block diffusion, motivating a criterion that does not depend on a predetermined position.

3.3 Using queries to identify important key directions

To avoid selecting protected tokens by position, we instead assess how errors in each key direction affect attention scores. These scores are query–key dot products, so a key error of a given magnitude has a larger effect along directions with larger query projections. Queries therefore provide a way to assess which directions need accurate storage without first locating a sink token.

The SDAR measurements show why key statistics alone can underestimate a direction's importance to attention. We project pre-RoPE keys and queries onto the same key principal directions and measure their energy using squared projection values. In the analyzed layers and heads, the lower half of directions ranked by key energy accounts for less than 1% of key energy but 64% of query energy (Figure 6). Directions with little key energy can therefore have large query projections, making

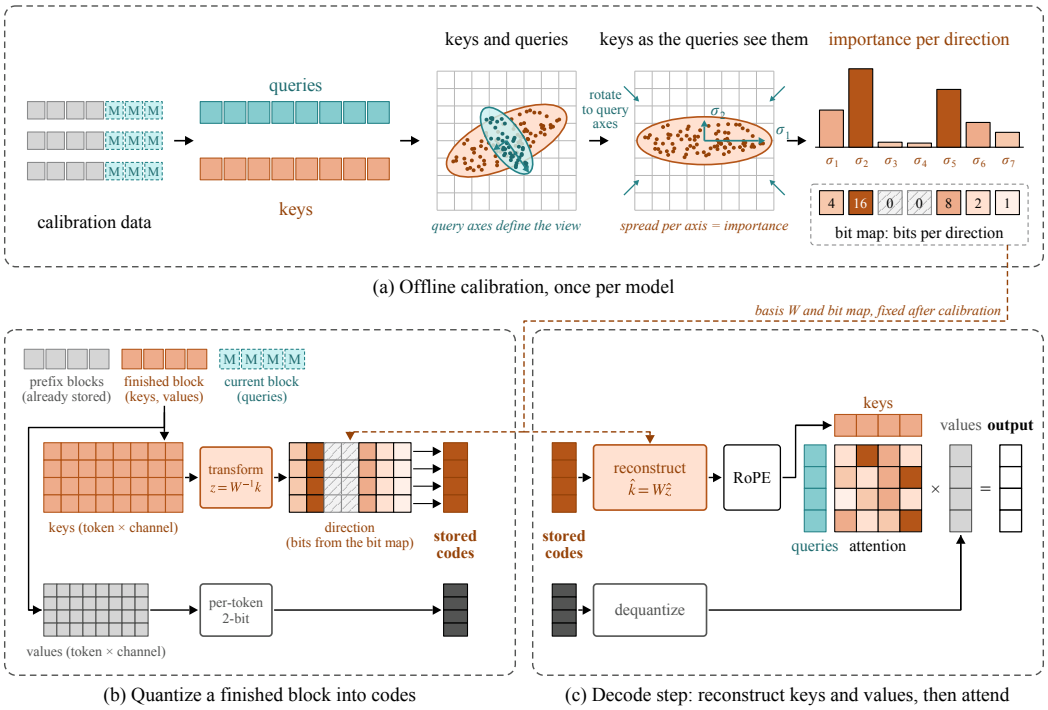


Figure 3: QuMix compresses the KV cache while preserving key information important to attention. **a)** One-time calibration uses queries and keys to compute a key basis and assign a bit width to each direction. **b)** Completed token groups are projected, quantized, and stored; values use per-token 2-bit quantization. **c)** During attention, keys are reconstructed before RoPE is applied, and values are dequantized.

errors along them important to attention scores. This mismatch motivates allocating precision by the effect of key errors on query–key scores. Section 4 develops this criterion into a basis and bit allocation that prioritize important key information without assuming a sink position.

4 QuMix: Query-weighted KV-cache quantization for dLLMs

QuMix uses calibration queries and keys to choose key directions for quantization and determine how many bits each direction needs. It constructs a query-weighted basis that lets us measure how errors in each direction affect attention scores. These measurements guide bit allocation: under a fixed storage budget, directions whose errors have a larger effect receive more bits. The basis and bit allocation are fixed for each layer and KV head and reused across requests and denoising steps, without changing model weights. Figure 3 shows how QuMix uses this basis and allocation to compress the cache and compute attention.

4.1 Finding key directions that matter to queries

To decide how many bits each key direction needs, we measure how errors in that direction change attention scores. We use queries and keys before RoPE, matching the stored key representation (Hooper et al., 2024). For a pre-RoPE query q, the key reconstruction error $\Delta k = \hat{k} - k$ changes their dot product by $\Delta s = q^\top \Delta k$. Averaging the squared change across queries gives

$$\mathbb{E}_q[(\Delta s)^2] = \Delta k^\top \bar{Q} \Delta k, \qquad \bar{Q} = \mathbb{E}[qq^\top]. \tag{1}$$

The matrix $\bar{Q}$ determines how strongly key errors in each direction affect the scores (Zhou et al., 2026; Fernández-Menduiña et al., 2026). This pre-RoPE measure approximates attention-score error without modeling relative positional rotations. We use it to construct a basis in which the error can be evaluated separately for each direction.

We construct this basis by weighting keys according to the queries before finding their principal directions. From calibration queries and keys, we estimate $\bar{Q}$ and $\bar{K} = \mathbb{E}[kk^\top]$. Factoring $\bar{Q} = LL^\top$ gives $\|L^\top \Delta k\|_2^2 = \Delta k^\top \bar{Q} \Delta k$: reconstruction error after applying $L^\top$ is exactly the query-weighted error in Equation 1. The principal directions of the weighted keys are then given by

$$L^\top \bar{K} L = US^2U^\top, \qquad S = \mathrm{diag}(\sigma_1, \ldots, \sigma_D). \tag{2}$$

The columns of U identify the principal directions after query weighting, and σ_d^2 measures the key energy along each. These directions therefore reflect both the information stored in the keys and how the queries use it.

The values along these directions can have different magnitudes. We normalize them using S to compare quantization errors on the same scale and express each key in the resulting basis W:

$$W = L^{-\top} US, \qquad z = W^{-1}k, \qquad \hat{k} = W\hat{z}. \tag{3}$$

Here z_d is the key's coefficient along direction d, the dth column of W. The cache stores quantized coefficients $\hat{z}$; multiplying by W reconstructs the key. The query-weighted error is now $\sum_d \sigma_d^2(\hat{z}_d - z_d)^2$, a sum of errors from individual directions. For coefficient errors of the same size, a direction with larger σ_d has a larger effect on the scores and therefore needs more accurate storage.

4.2 Choosing precision under a storage budget

The basis tells us how strongly errors in each direction affect attention scores. To assign bit widths, we also estimate how much those errors decrease when a direction receives more bits. For normalized coefficients, a scalar-quantization approximation makes direction d's weighted error proportional to $\sigma_d^2 2^{-2b_d}$. If bit widths could take continuous values, minimizing the sum of these errors under a fixed total bit budget would give $b_d = \log_2 \sigma_d + c$. This relation assigns more bits to directions with larger σ_d. We round the resulting bit widths to those supported by the quantizer:

$$\begin{aligned} b_d &= \mathrm{nearest}_{\mathcal{B}}(\log_2 \sigma_d + c), \\ \mathcal{B} &= \{0, 1, 2, 4, 8, 16\}. \end{aligned} \tag{4}$$

Here $\mathrm{nearest}_{\mathcal{B}}$ rounds to the closest supported width. We adjust c by binary search for each layer and KV head to match the storage budget, including scale and offset metadata, and report the storage used by the resulting allocation. The error approximation is not exact at zero or one bit, so it motivates the allocation rule rather than guaranteeing an optimal discrete allocation. A direction assigned zero bits is omitted and reconstructed as zero, so the allocation determines both which directions are stored and their precision. Section 5.4 evaluates the resulting allocation.

The fixed allocation is applied as completed tokens enter the cache. Once 32 tokens are available, their key coefficients are quantized together at the assigned widths, sharing an 8-bit scale and an 8-bit offset per quantized direction. Values use per-token 2-bit quantization with 8-bit scales and offsets. The active block and incomplete groups remain at full precision. Appendix C gives the quantization and storage details.

4.3 Reading the compressed cache efficiently

The compressed representation saves storage, but reading it requires unpacking coefficients, reconstructing keys with W, and applying RoPE. Writing the reconstructed keys to GPU global memory before attention would add a full-cache write and read at every denoising step, reducing the benefit of storing fewer bits. Block diffusion repeats this cost whenever the active block attends to its completed history.

Our attention kernel combines key reconstruction with attention computation. It loads compressed token groups, unpacks their coefficients, reconstructs the keys and applies RoPE in registers and shared memory, then computes attention scores. Directions assigned zero bits require neither storage nor reconstruction, so the same allocation that reduces cache size also reduces the work of reconstructing keys. Values are dequantized as the attention output is formed. Each completed group is compressed once and its packed representation is reused by later reads.

This design lets attention use the smaller cache without writing the full reconstructed cache to global memory. Section 5.3 measures the resulting batch capacity and decoding speed, and Appendix E.2 gives the kernel design.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Models and workloads. We evaluate cache quantization on LongBench (Bai et al., 2024), which covers question answering, summarization, and code completion. The main comparison uses SDAR-8B-Chat (Cheng et al., 2026) and Nemotron-Labs-Diffusion-8B (Fu et al., 2026); Appendix D.1 adds TraDo-8B-Instruct (Wang et al., 2026). Each task uses 150 examples with inputs capped at 4K tokens, and all methods share the evaluation examples and decoding settings within each model. QuMix is calibrated on WikiText training text (Merity et al., 2016). Table 4 gives the settings.

Comparisons. We compare quality and cache storage against an unquantized reference, FP8, per-token 2-bit quantization, and implementations of KIVI, SVDq, and MixKVQ (Liu et al., 2024b; Hong et al., 2025; Zhang et al., 2026b). For each baseline, we also adjust QuMix's allocation to match its cache cost, separating the effect of the quantizer from differences in storage. Reported KV bits include quantization metadata and unquantized residuals. Appendix D.1 reports full per-task results, including KVQuant (Hooper et al., 2024); Appendix A describes the method-specific settings.

5.2 QUALITY UNDER KV-CACHE COMPRESSION

Table 1: LongBench performance and KV-cache cost. Avg. is the mean task score; KV is the estimated average bits per cache element. QuMix (same KV) rebuilds QuMix's allocation at each baseline's KV bits (Appendix A.3). Bold marks the best score within each matched pair.

	SDAR-8B			Nemotron-8B		
Method	KV	Avg.	**QuMix (same KV)**	KV	Avg.	**QuMix (same KV)**
16-bit reference	16.00	47.2	–	16.00	33.2	–
FP8 (E4M3)	8.15	**47.1**	**47.1** (±0.0)	8.15	**33.0**	32.9 (−0.1)
MixKVQ	3.28	**47.1**	46.8 (−0.3)	3.28	32.4	**32.9** (+0.5)
KIVI 2-bit	2.94	**46.7**	46.6 (−0.1)	2.94	32.2	**32.8** (+0.6)
SVDq-1.5 + 2-bit V	2.38	41.2	**46.4** (+5.2)	2.38	31.3	**32.7** (+1.4)
INT2 per-token	2.30	3.7	**46.3** (+42.6)	2.30	21.5	**32.8** (+11.3)
KIVI 1-bit	2.15	11.6	**46.1** (+34.5)	2.18	11.9	**33.0** (+21.1)
SVDq-1.0 + 2-bit V	1.88	32.6	**45.0** (+12.4)	1.88	13.1	**32.5** (+19.4)
QuMix (ours)	**1.69**	–	**44.8**	**1.69**	–	**31.5**
QuMix (ours)	**1.55**	–	**39.1**	**1.55**	–	**30.5**

Comparable quality at lower storage cost. At 1.99 average KV bits, QuMix retains nearly all of the unquantized score on both SDAR and Nemotron while using less storage than KIVI 2-bit and MixKVQ (Appendix D.1). Its score is close to these methods on SDAR and higher on Nemotron. To compare the quantizers at equal storage, the QuMix (same KV) columns in Table 1 match each baseline's cache cost. Under this control, QuMix scores higher than every baseline at 2.38 bits or fewer; at larger budgets, the methods remain close to the unquantized reference.

Quality as compression increases. Further compression affects tasks differently, which the average score can obscure. At 1.69 and 1.55 bits, QuMix still outperforms SVDq's 1.88-bit setting on both models, but the loss depends on the task: SDAR's QA and summarization scores remain more stable, whereas code completion degrades more at the lowest precision (Figure 4). The usable compression level therefore depends on the workload.

Longer inputs with a fixed calibration. To test whether the same representation remains useful as the cache grows, we evaluate RULER with calibration and bit assignments fixed at the 4K settings (Appendix D.4).

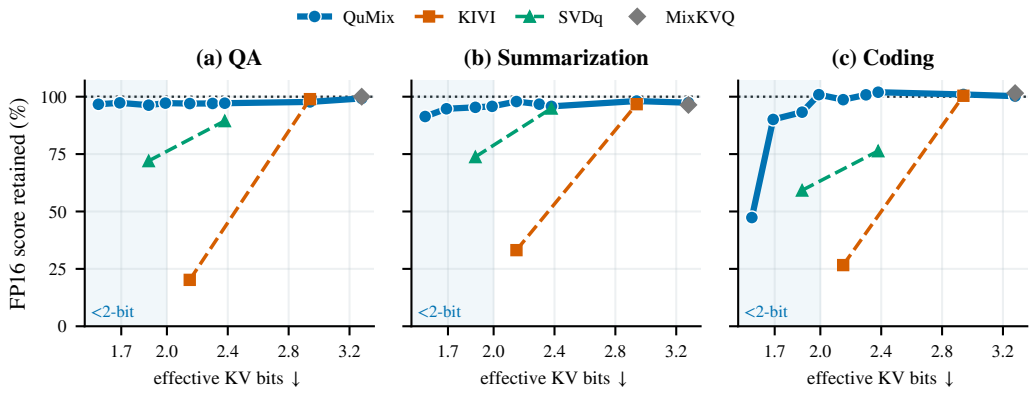


Figure 4: With QuMix, QA and summarization tolerate lower precision better than code completion. SDAR scores are relative to the unquantized reference (100%). The QuMix curve also includes the same-KV settings of Table 1 (1.88–3.28 bits).

Table 2: RULER accuracy across input lengths (13-task average). Bold marks the better quantized result.

	SDAR-8B				Nemotron-8B		
KV (bits)	4K	8K	16K	32K	4K	8K	16K
16-bit	94.5	94.1	88.8	76.1	87.6	78.6	71.9
KIVI 2-bit (2.94)	92.7	90.8	80.6	67.8	62.1	56.8	43.7
QuMix (1.99)	**93.4**	**92.4**	**84.2**	**70.5**	**85.7**	**72.6**	**63.9**

QuMix scores above KIVI at every evaluated length while using fewer KV bits (Table 2). Both models also lose accuracy with a 16-bit cache as inputs grow, while the gap between QuMix and the 16-bit reference widens: cache quantization adds to the models' loss at longer lengths.

Composition with weight quantization. We also keep the cache calibration and allocation fixed when applying 4-bit GPTQ to the model weights. Cache quantization costs 2.2 points on the five-task average with 4-bit weights, compared with 1.6 points with 16-bit weights (Appendix D.3).

5.3 Memory capacity and decoding efficiency

We test whether the smaller cache supports more requests and faster decoding, comparing 1.55-bit QuMix with BF16 FlashAttention4 (FA4) on SDAR-8B and one RTX PRO 6000. Each request has a 32K-token input and generates 64 output positions; both methods complete prefill and decoding. We compare speed at a common batch size and throughput at each method's largest batch. The cache uses the same 8-bit scale format as the quality evaluation. Appendix E gives the timing protocol, and Appendix E.2 reports kernel validation.

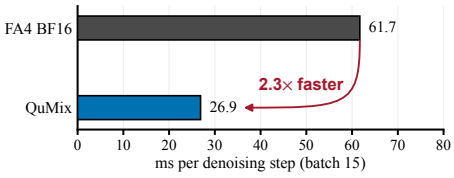


Figure 5: Denoising step time at batch 15.

A smaller cache supports larger batches. At batch 15, QuMix reduces KV-cache storage from 67.5 to 6.6 GiB (Table 13), allowing the maximum batch to increase from 15 to 157 under the same GPU memory limit (Figure 1a).

Faster decoding at the same batch size. The gain is not limited to fitting larger batches. At batch 15, a denoising step is $2.3\times$ faster with QuMix (Figure 5), and the full decoding phase takes 2.23 s versus 4.96 s with BF16, including cache updates and sampling (Table 13).

Throughput at maximum batch size. Using the additional capacity raises aggregate decode throughput from 194 to 754 output positions per second when each method runs at its largest batch, a $3.9\times$ gain (Table 13). This comparison combines the faster compressed-cache reader with the larger batch enabled by reduced storage. The gain for a complete request is smaller because both methods retain the same sequential BF16 prefill, which dominates total time in this workload.

5.4 Ablation

Table 3: Key-basis comparison on SDAR at approximately 1.54 average KV bits (TriviaQA F1).

Key basis	TriviaQA	Δ
query-weighted (QuMix)	88.8	–
original key channels	64.5	-24.3

QuMix combines a change of key basis with mixed precision. To test whether mixed precision on the original key channels is sufficient, we replace the query-weighted basis with those channels while keeping the calibration text and allocation rule fixed and matching the storage budget. Appendix D.2 gives the allocation settings for this controlled comparison.

At approximately 1.54 average KV bits, using the original channels reduces SDAR's TriviaQA score from 88.8 to 64.5 (Table 3), whereas the same replacement costs about one point at 1.88 bits. At the tighter tested budget, mixed precision on the original channels therefore falls well short of the same allocation rule in the query-weighted basis.

6 Conclusion

We present, to our knowledge, the first study of KV-cache quantization for block-diffusion language models. Our findings show that accurate compression below two bits depends on identifying which key information needs higher precision, while key variance and first-token protection can miss information important to attention. QuMix addresses this problem by using calibration queries and keys to construct a query-weighted basis and allocate bits according to how key errors affect attention scores, without assuming a sink position. Across three block-diffusion models, it retains nearly all of the original LongBench score at about two bits per cache element. Together with a kernel that computes attention directly over the compressed cache, QuMix reduces the cost of storing and repeatedly attending to completed blocks during denoising, enabling larger batches and faster block-diffusion inference.

AI use statement

The research idea originated with the authors. We used generative AI tools to assist with related-work searches, portions of the experimental code, drafting and revising manuscript text, and figure preparation. The authors reviewed the AI-assisted material, verified cited sources, and checked reported results against experimental outputs. The authors take full responsibility for the final manuscript, code, and findings.

Reproducibility statement

The query-weighted basis and mixed-precision allocation are described in Section 4, with implementation details in Appendix C. Appendix A specifies the calibration and evaluation settings, and Appendix E documents the CUDA implementation, numerical validation, and timing procedures. The supplementary material includes the QuMix implementation, calibration scripts, evaluation scripts for QuMix and the 16-bit reference, and CUDA serving kernels. Accompanying documentation provides environment setup and execution instructions.

REFERENCES

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

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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, 2024. doi: 10.18653/v1/2024.acl-long.172.

Shuang Cheng, Yihan Bian, Dawei Liu, Yuhua Jiang, Yihao 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. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 22058–22075, 2026. doi: 10.18653/v1/2026.findings-acl.1110.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Samuel Fernández-Menduiña, Amir Ziashahabi, Eduardo Pavez, Antonio Ortega, and Salman Avestimehr. Spend bits where queries look: KV cache vector quantization with attention-preserving transforms, 2026. URL https://arxiv.org/abs/2608.04074.

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*, 2023. URL https://arxiv.org/abs/2210.17323.

Yonggan Fu, Lexington Whalen, Abhinav Garg, Chengyue Wu, Maksim Khadkevich, Nicolai Oswald, Enze Xie, Daniel Egert, Sharath Turuvekere Sreenivas, Shizhe Diao, Chenhan Yu, Ye Yu, Weijia Chen, Sajad Norouzi, Jingyu Liu, Shiyi Lan, Ligeng Zhu, Jin Wang, Jindong Jiang, Morteza Mardani, Mehran Maghoumi, Song Han, Ante Jukić, Nima Tajbakhsh, Jan Kautz, and Pavlo Molchanov. Nemotron-Labs-Diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding, 2026. URL https://arxiv.org/abs/2607.05722.

Xiangming Gu, Tianyu Pang, Chao Du, Qian Liu, Fengzhuo Zhang, Cunxiao Du, Ye Wang, and Min Lin. When attention sink emerges in language models: An empirical view. In *International Conference on Learning Representations*, 2025.

Yankun Hong, Xing Li, Hui-Ling Zhen, Xianzhi Yu, Wulong Liu, and Mingxuan Yuan. SVDq: 1.25-bit and 410x key cache compression for LLM attention. *arXiv preprint arXiv:2502.15304*, 2025.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems*, 2024.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2404.06654.

Fengfeng Liang, Yuechen Zhang, and Jiaya Jia. RoPE-aware bit allocation for KV-cache quantization, 2026. URL https://arxiv.org/abs/2606.24033.

Ruikang Liu, Haoli Bai, Haokun Lin, Yuening Li, Han Gao, Zhengzhuo Xu, Lu Hou, Jun Yao, and Chun Yuan. IntactKV: Improving large language model quantization by keeping pivot tokens intact. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 7716–7741, 2024a. doi: 10.18653/v1/2024.findings-acl.460.

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. In *International Conference on Machine Learning*, 2024b.

Binglei Lou, Haoran Wu, Kevin Lau, Gregor MacDonald, Jiayi Nie, Yao Lai, Can Xiao, Xuan Guo, Jianyi Cheng, Rika Antonova, Robert Mullins, and Aaron Zhao. NPU design for diffusion language model inference, 2026. URL https://arxiv.org/abs/2601.20706.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016. URL https://arxiv.org/abs/1609.07843.

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. In *Advances in Neural Information Processing Systems*, 2025. doi: 10.52202/085713-1689.

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. doi: 10.18653/v1/2026.findings-acl.2145.

Yuerong Song, Xiaoran Liu, Ruixiao Li, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. Sparse-dLLM: Accelerating diffusion LLMs with dynamic cache eviction. *arXiv preprint arXiv:2508.02558*, 2025. URL https://arxiv.org/abs/2508.02558.

Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. Revolutionizing reinforcement learning framework for diffusion large language models. In *International Conference on Learning Representations*, 2026.

Haocheng Xi, Harman Singh, Yuezhou Hu, Coleman Hooper, Rishabh Tiwari, Aditya Tomar, Minjae Lee, Wonjun Kang, Michael Mahoney, Chenfeng Xu, Kurt Keutzer, and Amir Gholami. LoSA: Locality aware sparse attention for block-wise diffusion language models. *arXiv preprint arXiv:2604.12056*, 2026.

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

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 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.

Ted Zadouri, Markus Hoehnerbach, Jay Shah, Timmy Liu, Vijay Thakkar, and Tri Dao. FlashAttention-4: Algorithm and kernel pipelining co-design for asymmetric hardware scaling. *arXiv preprint arXiv:2603.05451*, 2026.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. TurboQuant: Online vector quantization with near-optimal distortion rate. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/5c802ef38ab6e366c2ea06eee554c088-Abstract-Conference.html.

Jiamu Zhang, Liang Wu, Kelly Wan, Hanjie Chen, and Liangjie Hong. SPECTRA: Pushing the KV cache beyond the 2-bit cliff via spectral transform coding, 2026a. URL https://arxiv.org/abs/2608.07915.

Tao Zhang, Ziqian Zeng, Hao Peng, Huiping Zhuang, and Cen Chen. MixKVQ: Query-aware mixed-precision KV cache quantization for long-context reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7189–7204, 2026b. doi: 10.18653/v1/2026.acl-long.326.

Zhongzhu Zhou, Donglin Zhuang, Jisen Li, Ziyan Chen, Shuaiwen Leon Song, Ben Athiwaratkun, and Xiaoxia Wu. OSCAR: Offline spectral covariance-aware rotation for 2-bit KV cache quantization, 2026. URL https://arxiv.org/abs/2605.17757.

Fei Zuo, Zikang Zhou, Hao Cong, Xiaoyan Xi, and Ho Fai Leung. RateQuant: Optimal mixed-precision KV cache quantization via rate-distortion theory, 2026. URL https://arxiv.org/abs/2605.06675.

A Evaluation settings and baseline implementations

A.1 Models, calibration, and scoring

All quantizers share the examples, chat template, decoding policy, and scoring procedure within each model. Tables 4 and 5 specify the settings, metrics, and output limits. TraDo uses SDAR's block-four decoding settings rather than its official stochastic configuration.

Table 4: Evaluation settings. Decoding is shared across quantizers within each model; the lower block specifies QuMix calibration and storage.

Setting	SDAR	Nemotron
Examples per task	150	150
Maximum input tokens	4,096	4,096
Diffusion block size	4	32
Denoising policy	4 steps; static low-confidence	Native; confidence threshold 0.9
Sampling	Top-$k = 1$, temperature 0.1	Greedy
QuMix settings		
Calibration	64×512 tokens	64×512 tokens
Key quantization group (tokens)	32	32
Key bit-width set	$\{0, 1, 2, 4, 8, 16\}$	$\{0, 1, 2, 4, 8, 16\}$
Value precision	2 bits	2 bits

Table 5: LongBench task metrics and generation limits. Native stopping can end outputs earlier; block alignment can extend the configured limit by a few positions.

Task	Metric	Configured output tokens
Qasper	QA F1	128
TriviaQA	QA F1	32
HotpotQA	QA F1	32
MultiFieldQA-en	QA F1	64
NarrativeQA	QA F1	128
QMSum	ROUGE-L	512
MultiNews	ROUGE-L	512
SAMSum	ROUGE-L	128
LCC	Code similarity	64
RepoBench-P	Code similarity	64

Nemotron output processing. We trim Nemotron outputs at the first trailing `</think>` before scoring, using the same rule for all methods. SDAR and TraDo do not emit this marker.

A.2 Baseline settings and calibration

We apply local implementations of KIVI, KVQuant, SVDq, and MixKVQ to completed cache states, keeping the active block exact.

Reference quantizers. FP8 stores keys and values in E4M3 with one FP16 absolute-maximum scale per token and head. INT2 per-token stores both at two bits with per-token scales and offsets. Both seal complete 32-token groups and keep the incomplete group exact, giving estimated costs of 8.15 and 2.30 bits per cache element, respectively.

KIVI, SVDq, and MixKVQ. KIVI 2-bit uses a 128-token residual window; the one-bit comparison retains only incomplete groups. SVDq fits its key basis to the prefill cache and leaves generated

keys unquantized. MixKVQ is adapted to the reported budget. Its nominal storage cost is 3.28 bits per cache element in the main comparison.

KVQuant calibration and selection. KVQuant is reported in the full results. Its SDAR row uses the best of three calibration draws for Qasper, TriviaQA, and SAMSum; other cells use the recorded calibration. Nemotron and TraDo each use one model-specific calibration. Sparse storage uses the nominal 1% budget on SDAR and TraDo and observed outlier rates on Nemotron.

A.3 Cache cost and matched-storage comparisons

Reported KV bits are analytical estimates at a 4,100-token reference length and include quantization metadata and uncompressed residuals. The SVDq estimate also includes its prefill basis and increases with the unquantized generated suffix. Physical GPU memory, including workspaces, is measured separately in Appendix E. The main quality evaluation reconstructs quantized states before attention; the packed implementation is checked separately in Appendix E.2.

QuMix at the same KV cost. The QuMix (same KV) columns of Table 1 keep the calibration, basis, allocation rule, and two-bit values fixed, adjusting per-head offsets to match each baseline's analytical cache cost. Realized costs are at most 0.003 bits below their targets, except for FP8 on Nemotron (8.13 versus 8.15 bits). Evaluation and output processing follow the same protocol as the main comparison.

B Supporting attention analysis

B.1 TriviaQA evaluation protocol

Figures 1b and 2a use the first 150 LongBench TriviaQA examples with a 4,096-token input cap. The restoration experiment also evaluates Qwen3, using greedy decoding with thinking disabled and a 32-token output cap; SDAR follows Table 4. Each model uses its own chat template. Quantizer settings follow Appendix A; the QuMix curve uses the three settings in Table 6. Reported bits include metadata, whereas KIVI's one-/two-bit names refer to quantized element precision.

B.2 Restoring early keys

After SVDq reconstructs prefill keys, we restore the first key or first four keys to their exact pre-RoPE values. The basis is fitted on all prefill keys, values remain at two bits, and generated keys remain unquantized. The four-key intervention covers SDAR's first block and tests this group jointly. Figure 2a uses its own unprotected and BF16 runs as references.

B.3 Attention maps and sink positions

We measure attention from the last 256 prompt positions on 16 TriviaQA inputs, using causal attention for Qwen3 and block-causal attention with blocks of four for SDAR. Averaged over all 36 layers and 32 heads, the first token receives 35.8% of attention in Qwen3 but only 0.025% in SDAR. Thus, weak first-token attention in SDAR extends beyond the head shown in Figure 2b.

For the maps, we select the same layer and head index with the largest first-token attention difference (layer 13, head 12) and average attention among the first 12 prompt tokens across these inputs. These results concern SDAR's block-causal cache; a separate probe of Nemotron's native decoder finds a first-token sink under its causal cache construction.

B.4 Key and query energy

Figure 6 compares pre-RoPE keys and queries in SDAR layers 8, 18, 26, and 34. Keys come from a forward pass over GSM8K text (Cobbe et al., 2021); queries come from prefill and denoising calls on four GSM8K questions. We project both onto the same key principal directions and measure squared projection values, ranking directions by the key second moment. Key energy is pooled

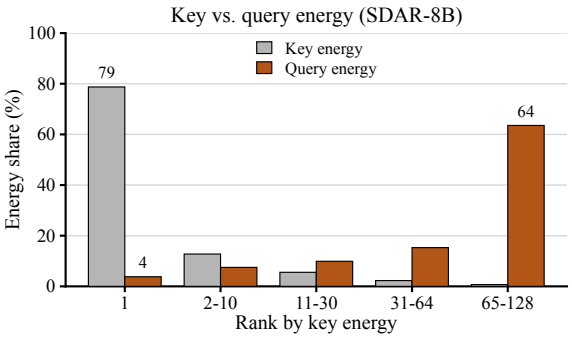


Figure 6: Key and query energy along SDAR's key principal directions, ranked by key energy.

across layers and heads; query second moments are normalized by their traces before projection and pooling. Each energy series is grouped by key rank and normalized to sum to 100%.

C Method details

C.1 Query-weighted basis

We collect uncentered query and key moments during prefill, denoising, and block commits. Query heads sharing a KV head contribute to its query moment. For the basis in Equation 3, the inverse is $W^{-1} = S^{-1}U^\top L^\top$. The implementation factors $C = \bar{Q} + \epsilon I = LL^\top$. Before numerical flooring, nonsingular moments give $\mathbb{E}[zz^\top] = I$ and $W^\top C W = S^2$. Thus, for $\Delta z = \hat{z} - z$, the regularized query-weighted error is

$$\Delta k^\top C \Delta k = \Delta z^\top S^2 \Delta z = \sum_d \sigma_d^2 (\Delta z_d)^2.$$

The basis is generally not orthogonal: encoding uses W^{-1}, not $W^\top$. The reference implementation specifies the numerical regularization and eigenvalue flooring.

The pre-RoPE metric averages query sensitivity and does not model every relative position: a key error at position s affects a query at t through $q^\top \mathcal{R}_t^\top \mathcal{R}_s \Delta k$. We reconstruct keys before applying their positional rotations at inference.

C.2 Precision allocation

We find the allocation offset by binary search per layer and KV head, rounding midpoint ties to the lower supported bit width. The main table first sets effective key budgets using $\{0, 2, 4, 8, 16\}$ at payload rates of 1.5, 1.0, and 0.75 bits, then matches those costs with $\{0, 1, 2, 4, 8, 16\}$. Controlled experiments target effective key costs directly. Reported costs use the realized discrete allocation.

C.3 Coefficient and metadata quantization

For a retained coordinate with group minimum m and maximum M, affine quantization at $0 < b < 16$ uses $s = \max(M - m, \delta)/(2^b - 1)$ and reconstructs $m + s\ \mathrm{clip}(\mathrm{round}((z - m)/s), 0, 2^b - 1)$. We use $\delta = 10^{-8}$ to avoid a zero scale. Dropped coordinates reconstruct as zero; values use the same rule across each token's head channels.

Scales and offsets are stored with eight bits each. The first 256 tokens set per-head, per-direction ranges for $\log_2 s$ and m, widened by one octave and 25%, respectively. Later groups quantize to 256 levels in these fixed ranges, clipping values outside them. Values follow the same rule per head with per-token scales and offsets. The simulation and packed kernel use this metadata format.

The simulation's 16-bit class bypasses scalar rounding and retains FP32 coefficients until key reconstruction, while charging the reference-width storage cost.

C.4 CACHE UPDATES AND STORAGE ACCOUNTING

Cache updates. All adapters use exact prefill attention and compress completed 32-token groups once; the active block and incomplete group remain exact. New compressed groups are visible during SDAR/TraDo's commit forward and after Nemotron's commit forward. Controlled comparisons use the same adapter.

Effective cache bits. For each direction quantized below 16 bits, a group of $G = 32$ tokens shares an 8-bit scale and an 8-bit offset. Reference-width directions are charged 16 bits without this metadata. With key dimension D, the compressed key rate is

$$r_K = \frac{1}{D}\sum_{d=1}^{D}\left(b_d + \frac{16}{G}\mathbf{1}[0 < b_d < 16]\right). \tag{5}$$

For cache length T and unquantized token counts f_K, f_V,

$$r_K^{\text{eff}} = r_K + \frac{f_K}{T}(16 - r_K), \qquad r_V^{\text{eff}} = r_V + \frac{f_V}{T}(16 - r_V), \qquad r_{KV}^{\text{eff}} = \frac{r_K^{\text{eff}} + r_V^{\text{eff}}}{2}.$$

The table convention uses $r_V = 2 + 16/D$ for two-bit values with eight-bit scales and offsets, and a mean incomplete-group tail of 15.5 tokens at $T = 4{,}100$. It omits the small per-request range headers (64 bits per head and retained quantized coordinate). Shared transforms, padding, headers, and workspaces are included in the separate physical memory measurements.

D FULL RESULTS AND ADDITIONAL QUALITY EVALUATIONS

D.1 FULL LONGBENCH RESULTS

Tables 6–8 expand the main comparison into per-task scores under the settings of Appendix A. KV denotes estimated effective cache bits, including metadata and uncompressed tokens; it is not measured GPU memory. Averages use unrounded task scores.

Table 6: Ten-task LongBench results on SDAR-8B. KV uses the $(K + V)/2$ convention.

Method	KV	Qasper	TriviaQA	HotpotQA	MFQA	NarrQA	QMSum	MultiNews	SAMSum	LCC	RepoB.-P	Avg.
16-bit reference	16	35.2	91.6	46.0	55.6	25.9	22.6	27.3	45.1	65.4	57.7	47.2
FP8 (E4M3)	8.15	34.4	91.6	45.9	56.0	25.4	22.3	26.8	45.2	65.0	58.6	47.1
MixKVQ	3.28	33.2	91.8	46.5	55.3	27.5	22.2	25.9	43.5	67.3	57.7	47.1
KIVI 2-bit	2.94	33.6	90.7	46.0	54.1	27.2	22.0	25.2	44.7	66.4	57.2	46.7
KVQuant nuq2-1%[†]	2.45	22.5	75.1	32.7	43.0	15.8	20.8	11.1	36.2	30.6	11.1	29.9
SVDq-1.5 + 2-bit V	2.38	30.8	85.5	38.7	50.9	21.6	22.1	25.7	42.4	52.0	42.0	41.2
INT2 per-token	2.30	1.1	4.1	2.2	4.1	2.6	2.0	3.4	1.4	12.2	4.3	3.7
KIVI 1-bit	2.15	5.7	19.3	6.3	12.3	7.8	11.6	5.6	14.3	18.7	14.1	11.6
QuMix (ours)	**1.99**	**37.1**	**91.8**	**42.2**	**51.7**	**24.4**	**22.3**	**25.2**	**43.5**	**66.4**	**57.8**	**46.2**
SVDq-1.0 + 2-bit V	1.88	20.2	68.6	33.9	45.1	15.4	20.8	24.3	25.0	43.2	29.7	32.6
QuMix (ours)	**1.69**	**34.9**	**92.0**	**43.7**	**54.0**	**22.9**	**21.8**	**24.1**	**44.1**	**62.2**	**48.7**	**44.8**
QuMix (ours)	**1.55**	**37.3**	**90.1**	**42.9**	**52.2**	**23.4**	**21.4**	**23.9**	**41.5**	**44.0**	**14.3**	**39.1**

[†] Best of three calibration draws on Qasper, TriviaQA, and SAMSum; the other cells use the recorded calibration.

Table 7: Ten-task LongBench results on Nemotron with its native block-32 decoder. Outputs are trimmed as specified in Appendix A.

Method	KV	Qasper	TriviaQA	HotpotQA	MFQA	NarrQA	QMSum	MultiNews	SAMSum	LCC	RepoB.-P	Avg.
16-bit reference	16	41.5	83.0	39.4	47.4	20.3	20.4	24.1	36.1	6.3	13.2	33.2
FP8 (E4M3)	8.15	41.1	83.2	39.2	47.6	20.4	20.4	23.9	35.6	6.0	12.7	33.0
MixKVQ	3.28	39.8	82.1	39.6	47.7	21.2	20.7	23.7	35.4	4.2	10.0	32.4
KIVI 2-bit	2.94	39.0	81.6	37.6	46.1	20.7	19.6	23.8	35.5	6.7	11.1	32.2
KVQuant nuq2-1%†	2.74	38.9	81.8	36.9	43.8	21.5	19.7	23.2	31.8	9.6	15.7	32.3
SVDq-1.5 + 2-bit V	2.38	38.5	80.2	35.6	42.1	20.7	20.5	24.7	30.9	5.9	14.2	31.3
INT2 per-token	2.30	22.9	44.0	26.0	28.2	17.0	15.5	18.3	11.0	12.2	19.9	21.5
KIVI 1-bit	2.18	5.2	19.3	11.9	10.8	6.4	11.2	5.1	6.0	20.1	22.6	11.9
QuMix (ours)	**1.99**	**39.8**	**82.0**	**38.3**	**48.1**	**21.7**	**20.9**	**24.0**	**35.2**	**4.7**	**14.4**	**32.9**
SVDq-1.0 + 2-bit V	1.88	10.8	22.5	14.5	15.1	6.4	11.7	15.6	6.0	14.5	13.4	13.1
QuMix (ours)	**1.69**	**39.2**	**75.6**	**31.0**	**46.0**	**20.0**	**20.7**	**24.3**	**34.6**	**7.6**	**16.3**	**31.5**
QuMix (ours)	**1.55**	**37.9**	**77.2**	**32.7**	**43.5**	**20.0**	**21.0**	**24.5**	**27.5**	**5.4**	**15.8**	**30.5**

†KVQuant uses one calibration draw and observed outlier rates; the reported 2.74 bits summarize per-task estimates of 2.697–2.824. KIVI 2-bit uses a 128-token residual. SVDq includes the prefill basis and leaves generated keys unquantized.

Table 8: Ten-task LongBench results on TraDo with the SDAR decoding settings of Appendix A.

Method	KV	Qasper	TriviaQA	HotpotQA	MFQA	NarrQA	QMSum	MultiNews	SAMSum	LCC	RepoB.-P	Avg.
16-bit reference	16	34.7	91.6	45.7	54.0	26.0	21.6	26.1	44.2	67.0	57.5	46.9
FP8 (E4M3)	8.15	35.4	91.6	46.0	54.1	26.9	21.8	26.7	42.9	66.4	58.6	47.0
MixKVQ	3.28	35.6	91.1	46.5	54.0	27.3	21.9	26.0	42.9	66.5	57.4	46.9
KIVI 2-bit	2.94	36.3	90.6	45.0	54.4	27.6	21.9	25.7	43.1	63.9	56.5	46.5
KVQuant nuq2-1%†	2.45	30.9	88.0	42.3	51.1	22.5	20.5	26.0	42.0	60.5	51.3	43.5
SVDq-1.5 + 2-bit V	2.38	32.3	90.1	37.9	50.9	19.6	21.5	24.7	43.5	50.4	44.7	41.5
INT2 per-token	2.30	1.8	3.7	0.6	4.1	1.6	2.4	4.4	1.5	12.5	3.8	3.6
KIVI 1-bit	2.15	6.4	22.7	7.2	12.1	9.1	10.8	7.0	15.1	18.9	14.9	12.4
QuMix (ours)	**1.99**	**36.7**	**90.3**	**42.9**	**53.2**	**25.9**	**21.9**	**25.5**	**43.1**	**64.0**	**59.1**	**46.3**
SVDq-1.0 + 2-bit V	1.88	19.1	77.9	32.3	46.5	16.4	20.3	24.6	29.5	41.8	31.1	33.9
QuMix (ours)	**1.69**	**39.1**	**91.3**	**44.1**	**55.2**	**25.4**	**21.3**	**24.8**	**41.9**	**59.2**	**50.0**	**45.2**
QuMix (ours)	**1.55**	**38.5**	**90.4**	**41.8**	**55.5**	**23.9**	**21.9**	**24.6**	**40.9**	**51.1**	**31.8**	**42.0**

†KVQuant uses a nominal 1% sparse budget and one model-specific calibration. MixKVQ uses the nominal 3.28-bit convention.

D.2 KEY BASIS AT TWO BUDGETS

We compare the query-weighted basis with original key channels at two storage budgets, using the same calibration text and allocation rule. The original-channel allocation is adjusted per head to match the budget; realized average KV costs differ by less than 0.002 bits (Table 9).

Table 9: SDAR TriviaQA F1 at two matched cache budgets (150 examples). KV includes metadata and the reference exact tail.

	Higher budget		Lower budget	
Key basis	KV bits	F1	KV bits	F1
Unquantized	16.000	91.6	16.000	91.6
Query-weighted (QuMix)	1.879	90.9	1.535	88.8
Original key channels	1.878	89.9	1.533	64.5

This experiment compares the complete query-weighted basis with the original key channels at matched storage.

Allocation and cost accounting. For original channels, importance is estimated as $\mathbb{E}[q_j^2]\mathbb{E}[k_j^2]$ from calibration prefill queries and cache-write keys. These controlled runs permit one-bit coefficients and use FP16 scales and offsets, separately from the default eight-bit-metadata settings in Table 1. Two-bit values with FP16 metadata cost 2.25 bits. The table includes the mean 15.5-token exact tail at the 4,100-token reference length.

D.3 Composition with weight quantization

To test whether cache compression remains useful after weight quantization, we apply GPTQ to SDAR-8B while retaining the KV calibration, basis, and allocation from the 16-bit-weight setting. GPTQ quantizes every linear layer to four bits with group size 128, using 128 WikiText-2 sequences of 2,048 tokens, damping 0.01, and no activation ordering (Frantar et al., 2023). Activations remain at 16 bits.

Table 10: QuMix with 4-bit weights on SDAR-8B. Weights are quantized by GPTQ with group size 128; five LongBench tasks with 150 examples each. The 16-bit-weight rows are from Table 6.

Weights	KV	MFQA	TriviaQA	HotpotQA	SAMSum	LCC	Avg.
16-bit	16-bit	55.6	91.6	46.0	45.1	65.4	60.7
16-bit	QuMix 1.99	51.7	91.8	42.2	43.5	66.4	59.1
GPTQ 4-bit	16-bit	54.3	90.9	44.3	44.3	64.8	59.7
GPTQ 4-bit	QuMix 1.99	53.3	89.6	40.9	41.4	62.1	57.5

D.4 Longer inputs with fixed calibration

The RULER evaluation tests retrieval, tracking, and aggregation as input length grows (Hsieh et al., 2024). We use the official generators with 50 examples per task and length, retaining the cache calibration and bit assignments from the 4K evaluation. Tables 11 and 12 provide the thirteen task scores behind Table 2. KV labels use the reference costs of Tables 6 and 7; 32K results are available for SDAR only.

SDAR was pre-trained on 4,096-token sequences; Nemotron uses 4,096-token pre-training and 16K fine-tuning. We therefore compare each quantizer with the 16-bit reference at the same input length.

The averages conceal substantial task differences. SDAR's 16K loss is largest on MK-3; at 32K its largest loss is on common-word extraction (CWE), where it also falls below KIVI. Nemotron shows large KIVI losses on the S-3 and MK-3 retrieval variants, while QuMix also incurs increasing losses on these tasks at longer lengths.

Table 11: RULER on SDAR-8B by input length, all thirteen tasks (50 examples each; accuracy). S-, MK-, MQ, MV: single-key, multi-key, multi-query, and multi-value needle retrieval; VT: variable tracking; CWE, FWE: common- and frequent-word extraction; QA-1, QA-2: question answering over SQuAD and HotpotQA passages.

Length	KV (bits)	S-1	S-2	S-3	MK-1	MK-2	MK-3	MQ	MV	VT	CWE	FWE	QA-1	QA-2	Avg.
4K	16-bit	100.0	100.0	100.0	100.0	98.0	100.0	99.0	86.0	100.0	99.8	83.3	90.0	72.0	94.5
	KIVI 2-bit (2.94)	100.0	100.0	100.0	100.0	94.0	92.0	99.0	90.0	99.6	95.0	85.3	86.0	64.0	92.7
	QuMix (1.99)	100.0	100.0	100.0	100.0	98.0	98.0	99.5	85.5	100.0	94.2	81.3	90.0	68.0	93.4
8K	16-bit	100.0	100.0	100.0	96.0	98.0	100.0	99.5	87.0	100.0	96.4	98.0	78.0	70.0	94.1
	KIVI 2-bit (2.94)	100.0	100.0	100.0	96.0	92.0	76.0	94.5	92.5	98.4	91.8	96.7	72.0	70.0	90.8
	QuMix (1.99)	100.0	100.0	98.0	96.0	96.0	94.0	97.5	87.5	98.4	91.0	96.7	74.0	72.0	92.4
16K	16-bit	100.0	100.0	100.0	94.0	90.0	70.0	97.5	91.0	98.8	80.0	97.3	74.0	62.0	88.8
	KIVI 2-bit (2.94)	100.0	100.0	94.0	92.0	72.0	18.0	91.0	88.5	95.6	71.0	97.3	74.0	54.0	80.6
	QuMix (1.99)	100.0	100.0	100.0	92.0	76.0	34.0	94.5	94.5	98.8	71.6	98.7	76.0	58.0	84.2
32K	16-bit	100.0	100.0	96.0	88.0	48.0	8.0	92.0	93.0	83.6	75.4	96.7	58.0	50.0	76.1
	KIVI 2-bit (2.94)	100.0	100.0	92.0	86.0	16.0	4.0	76.0	81.5	75.6	55.8	96.0	46.0	52.0	67.8
	QuMix (1.99)	100.0	100.0	98.0	80.0	42.0	4.0	85.0	91.0	82.4	34.0	92.7	58.0	50.0	70.5

Table 12: RULER on Nemotron-8B by input length, all thirteen tasks; columns as in Table 11.

Length	KV (bits)	S-1	S-2	S-3	MK-1	MK-2	MK-3	MQ	MV	VT	CWE	FWE	QA-1	QA-2	Avg.
4K	16-bit	94.0	96.0	56.0	100.0	96.0	66.0	95.5	88.5	100.0	99.0	99.3	86.0	62.0	87.6
	KIVI 2-bit (2.94)	56.0	52.0	6.0	46.0	74.0	10.0	63.5	63.5	98.8	94.8	97.3	82.0	64.0	62.1
	QuMix (1.99)	96.0	92.0	46.0	94.0	96.0	70.0	94.5	88.0	98.4	93.8	99.3	84.0	62.0	85.7
8K	16-bit	84.0	88.0	46.0	90.0	94.0	62.0	84.5	76.5	96.8	89.4	90.7	64.0	56.0	78.6
	KIVI 2-bit (2.94)	76.0	54.0	4.0	62.0	58.0	6.0	39.5	43.5	90.0	87.8	88.0	70.0	60.0	56.8
	QuMix (1.99)	86.0	80.0	34.0	82.0	92.0	32.0	67.5	69.5	94.0	86.6	90.7	74.0	56.0	72.6
16K	16-bit	82.0	76.0	46.0	82.0	76.0	58.0	74.0	65.5	98.4	75.4	88.0	62.0	52.0	71.9
	KIVI 2-bit (2.94)	46.0	34.0	0.0	14.0	42.0	0.0	29.5	30.0	86.8	70.6	94.7	66.0	54.0	43.7
	QuMix (1.99)	84.0	72.0	24.0	88.0	60.0	18.0	66.0	54.5	90.0	70.6	94.0	60.0	50.0	63.9

E CUDA IMPLEMENTATION, VALIDATION, AND EFFICIENCY

E.1 HARDWARE, WORKLOAD, AND TIMING

We measure SDAR-8B on one RTX PRO 6000 Blackwell Server Edition (SM120), with PyTorch 2.9.1+cu130 and a 90 GiB allocator budget. QuMix uses the 1.55-bit allocation; the BF16 reference uses FlashAttention4 (FA4) 4.0.0b30 with one sequence split per head.

The capacity workload repeats a 32,704-token prompt across requests with independent KV allocations, then generates 64 output positions in blocks of four (64 denoising and 16 commit calls). Runs continue after EOS to keep the work fixed. Both paths use sequential BF16 FA4 prefill, including packing time for QuMix, and CUDA-graph decoding on an idle GPU. Loading, compilation, warmup, and graph capture are excluded. Table 13 includes model operations, cache updates, and sampling; Table 14 measures denoising calls over 50 graph replays without sampling or the host loop.

E.2 PACKED-CACHE IMPLEMENTATION AND VALIDATION

The packed cache uses 32-token groups, mixed 1/2/4/8-bit key coefficients, two-bit values, and eight-bit scales and offsets. Zero-bit directions are omitted; the timed allocation has no 16-bit directions. The active block and incomplete group remain exact. The kernel combines key reconstruction, RoPE, and attention without writing a dense prefix to global memory. The supplementary code documents packing, numerical formats, and launch settings.

Numerical and task-quality validation. On 150 TriviaQA examples, the timed packed kernel scores 90.3 F1 versus 90.1 for the simulation (Table 6), using SDAR at batch one in eager mode. Attention outputs differ by at most 0.23% in relative ℓ_2 from a reference reconstructed from the same packed record, measured at layers 0, 17, and 35 in three decoding calls per example. This comparison checks implementation error separately from quantization error; maximum-batch timing uses the fixed-work protocol above.

E.3 CAPACITY AND DECODING PERFORMANCE

BF16 completes batch 15 and fails at 16; QuMix completes batch 157 and fails at 158. Table 13 reports medians of three batch-15 runs and the single maximum-batch QuMix run. Throughput divides completed output positions by decode time; total time also includes prefill.

Table 13: Serving time and memory on RTX PRO 6000. Cache is allocated KV storage; Peak is peak prefill allocation, including weights and workspaces.

Method	Batch	Prefill (s)	Decode (s)	Total (s)	Decode pos./s	Cache (GiB)	Peak (GiB)
FA4 BF16	15	50.0	4.96	55.0	194	67.5	86.1
QuMix	15	50.9	2.23	53.1	430	6.6	25.3
QuMix	157	531.1	13.32	544.5	754	69.2	88.5

Table 14: Median denoising-step latency (ms) at 32,704 input tokens. The last row uses each method's maximum batch (15 / 157).

Batch	FA4 BF16	QuMix	Speedup
1	51.2	14.9	3.4×
8	50.7	21.1	2.4×
15	61.7	26.9	2.3×
largest (15 / 157)	61.7	149.4	–

Baseline configuration. FA4's one-split configuration underutilizes the GPU at batches one and eight, so those speedups reflect the baseline configuration as well as compression. At the main comparison's batch 15, FA4 and PyTorch SDPA's flash and cuDNN backends have similar per-layer attention times (1.33–1.34 ms).