KVARN: VARIANCE-NORMALIZED KV-CACHE QUANTIZATION FOR IMPROVED REASONING IN LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling improves language-model reasoning but increases KV-cache memory during long-horizon generation. Existing KV-cache quantization methods are commonly evaluated in static prefill settings, which do not capture feedback from repeatedly attending to a quantized cache. We show that this feedback increases attention-output error over time and identify token-scale distortion as a major contributor to the largest key-quantization errors. We introduce KVarN, a calibration-free, linear KV-Cache quantization method to effectively suppress token scale errors and error accumulation. At an effective storage cost of 2.25 bits per element, KVarN achieves the best overall quality against strong quantization baselines on AIME 2024, MATH-500, HumanEval, and IFEval. It also reduces attention-output error under our pseudo-decode diagnostic. In end-to-end vLLM measurements across three models, KVarN improves decode throughput by 10–28% over the 16-bit KV-cache baseline; at a nominal 3-bit precision, it achieves 1.44–2.00× the throughput of the vLLM TurboQuant implementation. Code for the KVarN method is given in the supplementary.

1 INTRODUCTION

Recently, test-time scaling (OpenAI, 2024) has helped reasoning LLMs achieve new levels of competence in a variety of domains (Snell et al., 2025; Bi et al., 2025; Muennighoff et al., 2025; Shen et al., 2025). As the name implies, the premise of test-time scaling is an ever-increasing decoding length that supports this greater capability. Consequently, the efficient handling of the KV-Cache is becoming more and more relevant to achieve optimal reasoning per time trade-offs.

One avenue of attack on this memory bottleneck is KV-Cache quantization. Such methods try to assign fewer than 16 bits per element of the K and V matrices, often as few as 2 to 4. With effective reshaping and blocking strategies, linear methods like KIVI (Liu et al., 2024b) can already achieve good results, and codebook-based methods like TurboQuant (Zandieh et al., 2026) can further improve on them. Typical evaluation settings for these methods are large prefill problems, where a fixed, long context is quantized in parallel. Examples are Needle-in-a-Haystack (NiaH) (Kamradt, 2023), MultiQA (Talmor & Berant, 2019), or LongBench (Bai et al., 2024).

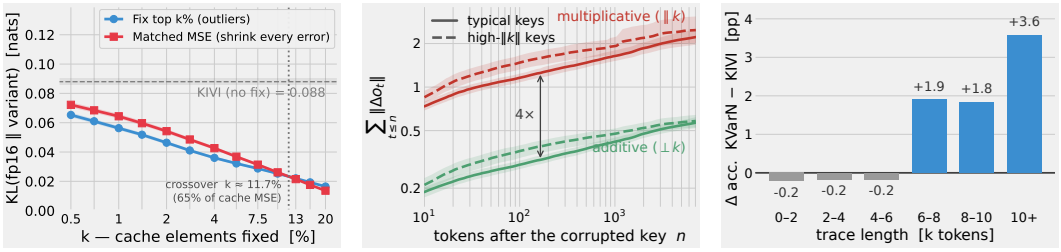
However, in long-form test-time scaling scenarios, the generated token sequence needs to be compressed on-the-fly as it is produced. As we will show in this paper, this leads to the accumulation of quantization error across decoding timesteps, because standard methods fail to preserve per-token scales during quantization.

One highly successful approach in weight quantization in recent years has been incoherence processing (Chee et al., 2023), e.g., by Hadamard rotation. We elucidate that this approach alone fails for KV-Cache compression, because it does not sufficiently address token scaling. We identify dual-scaling with Sinkhorn-based variance-normalization as an effective approach to further mitigate token scaling errors.

1.1 CONTRIBUTIONS

- **Error-accumulation diagnostic.** We introduce an efficient blockwise pseudo-decode evaluation that captures quantized-cache feedback and reveals error accumulation overlooked by static prefill evaluations.

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.



(a) Replacing the 1% worst K errors with high precision values improves end-to-end KL-divergence more than fixing an equal amount of MSE uniformly (see Fig. 13).

(b) Corrupting a single K vector with multiplicative rather than additive error causes greater attention-output error (Δo_t) over later tokens, even at matched error energy.

(c) KVarN improves MATH-500 accuracy over KIVI at 2-bit K/V precision on Qwen3-14B; binning gains by reasoning-trace length shows how this advantage varies with generation length (20-run average).

Figure 1: (a) Outlier errors dominate end-to-end degradation. (b) Multiplicative (magnitude) errors are more harmful than additive ones (see Eq. 2). (c) KVarN is constructed to prevent such errors and increasingly improves longer reasoning traces over baseline KV-Cache quantization.

- **Error decomposition.** We decompose key-quantization error into a token magnitude and an orthogonal residual, showing that scale distortion drives the largest errors and that correcting outliers disproportionately improves output quality.
- **Calibration-free KV-cache quantization.** We propose KV-Cache quantization with Variance Normalization (KVarN[1]): token- and channel-axis variance normalization as a targeted fix for token-scale errors and a synergistic mitigation of error accumulation with Hadamard rotation.
- **Generative quality at low precision.** At 2.25 bits per quantized cache element, including metadata, KVarN achieves the best mean scores among evaluated quantization baselines on AIME 2024, MATH-500, HumanEval, and IFEval. This enables longer functional reasoning traces in memory limited scenarios.
- **End-to-end efficiency.** In vLLM across three models, 2-bit KVarN improves decode throughput by 10–28% over the 16-bit KV-cache baseline; at nominal 3-bit precision, it delivers 1.44–2.00× the throughput of vLLM's TurboQuant implementation.

2 PRELIMINARIES

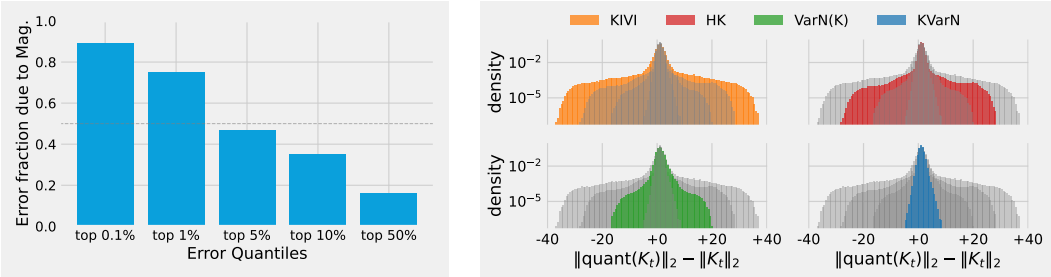
Unless otherwise mentioned, our experiments and figures use the KV-Cache of Qwen3-4B (Yang et al., 2025) ingesting wikitext-2. Generally K quantization is often considered to be more difficult than V quantization[2]; we will also focus more discussion on K because of this.

Following previous KV-Cache quantization work (Liu et al., 2024b), we process the KV-cache in the channel dimension per-head and in the token dimension in chunks (e.g. of size 128). Consequently, the base-tile to work with is a tile of shape (head-dim × token-chunk), e.g. for Llama3.1-8B in our setup a 128×128 tile. KIVI (Liu et al., 2024b) has shown that with round-to-nearest quantization, it is best to quantize the V matrix per token and the K matrix per channel. E.g., the K matrix is quantized to a set consisting of a low-precision matrix K_q, and two high-precision vectors, containing one element for each channel. One of these vectors is an offset (or zeropoint) $\vec{z}$ the other a scale $\vec{s}$. The dequantized approximation of K is K_{dq} obtained by

$$K_{dq} = (K_q + \vec{z}) \odot \vec{s}. \tag{1}$$

[1] kvarn, noun, Swedish: the grinding apparatus for substances such as grains, seeds, spices, coffee beans, KV-Caches, etc.

[2] As an illustration, in Llama3.1-8B $> 98\%$ of the top 5% quantization errors under the KIVI-scheme lie in a K-matrix.



(a) (b)

Figure 2: (a) What fraction of the top k% largest errors is due to magnitude rather than direction (decomposed as in Eq. 2). Large quantization errors in K are mostly due to incorrect token scaling. (b) Difference of per-token magnitude of quantized K to full-precision K matrix on Qwen3-4B under different 2-bit quantization methods (KIVI (Liu et al., 2024b), HK: Hadamard rotated K, VarN(K): Variance-Normalized K, and KVarN: our proposed method). KVarN effectively suppresses magnitude errors, which leads to better end-to-end test-time scaling.

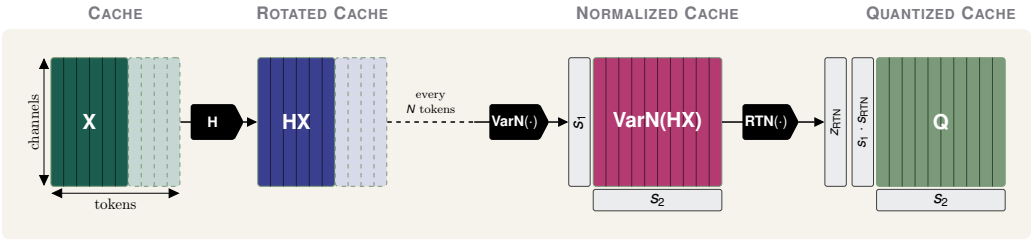


Figure 3: Schematic layout of KVarN. Every token is Hadamard-rotated in the channel dimension. After one block of tokens (e.g., 128) of generation, each block is variance-normalized in token and channel dimension, denoted by VarN(·). Finally, it is quantized with round-to-nearest (RTN). We store the standard scale and zero-point of the RTN, plus a second scale. At 2.3 average bits per element even with the second scale, KVarN outperforms or matches prior methods, see e.g. Tab. 3.

2.1 Key Ideas

The central finding of this paper is that a small fraction of large KV-cache quantization errors (e.g., the top 5%) causes most of the end-to-end degradation, while the many smaller errors are less consequential (Fig. 1). We identify distortion of per-token key magnitudes as a primary source of these outlier errors (Figs. 1, 2a). This diagnosis motivates KVarN, in which dual-axis variance normalization preserves per-token magnitudes and channel-wise Hadamard rotation suppresses channel outliers and decorrelates channels, each transform addressing a failure mode the other leaves open (Fig. 2b; App. D, Fig. 10). KVarN additionally suppresses the *error accumulation* we diagnose in Fig. 4 that arises during decoding, which static prefill evaluations do not capture (Figs. 4 and 5). Consequently, KVarN achieves state-of-the-art end-to-end KV-cache quantization performance on reasoning and instruction-following tasks (Tabs. 1, 2, and 3).

3 Methods

3.1 Magnitude Error Impact

To isolate token-scale distortion introduced by quantization, we decompose each dequantized key vector into a component parallel to its full-precision counterpart and an orthogonal residual. For a key vector K and sub-script $_{dq}$ indicating a dequantized tensor, we write

$$K_{dq} = (1+\delta)K + \eta, \qquad \delta = \frac{\langle K_{dq}, K\rangle}{\|K\|_2^2} - 1, \qquad \langle \eta, K\rangle = 0.$$

Here, δ is the relative multiplicative magnitude error, and η is the additive error orthogonal to K. This projection gives decomposition of the squared reconstruction error:

$$\underbrace{\|K_{dq} - K\|_2^2}_{E_T} = \underbrace{\delta^2 \|K\|_2^2}_{E_M} + \underbrace{\|\eta\|_2^2}_{E_A}. \tag{2}$$

E_M measures error along the original direction, while E_A measures error perpendicular to it.

In Fig. 2a, we report E_M/E_T across error quantiles on WikiText-2 and C4. Under the evaluated KIVI configuration, the largest errors are predominantly multiplicative.

This decomposition clarifies how key quantization affects attention. For a query vector Q, let K_t denote the key at token position t, and let d be the head dimension. The full-precision and quantized attention logits L satisfy

$$L_t = \frac{QK_t^\top}{\sqrt{d}}, \qquad L_{dq,t} = \frac{QK_{dq,t}^\top}{\sqrt{d}} = (1+\delta_t)L_t + a_t, \qquad a_t = \frac{Q\eta_t^\top}{\sqrt{d}}.$$

The multiplicative component therefore produces an attention logit perturbation $\delta_t L_t$ proportional to the original logit, while the additive component acts through the query's projection onto η_t.

With $p_t = \mathrm{softmax}(L)_t$, the change in attention probability is

$$\frac{p_{dq,t}}{p_t} = \frac{\exp(\delta_t L_t + a_t)}{\sum_j p_j \exp(\delta_j L_j + a_j)}.$$

For the multiplicative component this simplifies to $p_{dq,t}/p_t \propto \exp(\delta_t L_t)$, with a shared normalization factor across tokens. Token-scale errors can therefore strongly alter relative attention weights, particularly when they perturb large-magnitude logits.

If we perturb a single K vector with equal-energy additive or multiplicative error as defined here, we see that the additive error causes much less attention output damage over following tokens, Fig. 1b. We further see that injecting equal error energy into high magnitude keys, causes greater harm.

3.2 Evaluation of KV-Cache Quantization Error Accumulation

When the KV-Cache is quantized during decoding in deep models, a form of error accumulation occurs, see Fig. 4 for an illustration. Namely, when the attention in transformer block B_l at block-index l is computed with a *quantized* KV-Cache, this error influences the K and V matrices that the attention layer in the subsequent transformer block B_{l+1} produces. So the unquantized K and V matrices at B_{l+1} are already slightly wrong. Quantizing them introduces additional error, which in turn affects K and V at B_{l+2} and beyond. In this way, KV-Cache quantization errors can be accumulated across layers and eventually time-steps. This problem compounds as the generated sequence gets longer.

To directly measure the extent of this problem, we propose a fast decode-like LLM evaluation. Namely, we divide our full prefill sequence into blocks of size b. Whenever b tokens have passed through the model we are evaluating, we quantize the KV-Cache. Then, for all later tokens, we compute latent states *with this quantized KV-Cache* (as would be done at decoding time). We call this the 'pseudo-decode' setting. In Fig. 5 we show this error accumulation is specifically suppressed by KVarN, which we will introduce below. We evaluate free-running generation separately in Sec. 4.

3.3 KVarN: Variance-Normalized Quantization for KV-Cache

KVarN applies two transforms in sequence: Firstly, a channel-wise Hadamard rotation (Ashkboos et al., 2024), which equalizes channel-space outliers, followed by online variance normalization along both the channel and the token dimension, which preserves per-token magnitudes. Fig. 3 gives an overview. For the Hadamard transformation see Fig. 7: We rotate the K and V matrices in the channel dimension and reduces outliers there. The same transform does not carry over to the token dimension: it would have to be undone online at every position during dequantization (typically 128^2 operations per 128 tokens per channel). Token-axis normalization is therefore achieved by cheaper variance scaling instead: We apply scaling factors along both dimensions of the to-be-quantized tensor; for

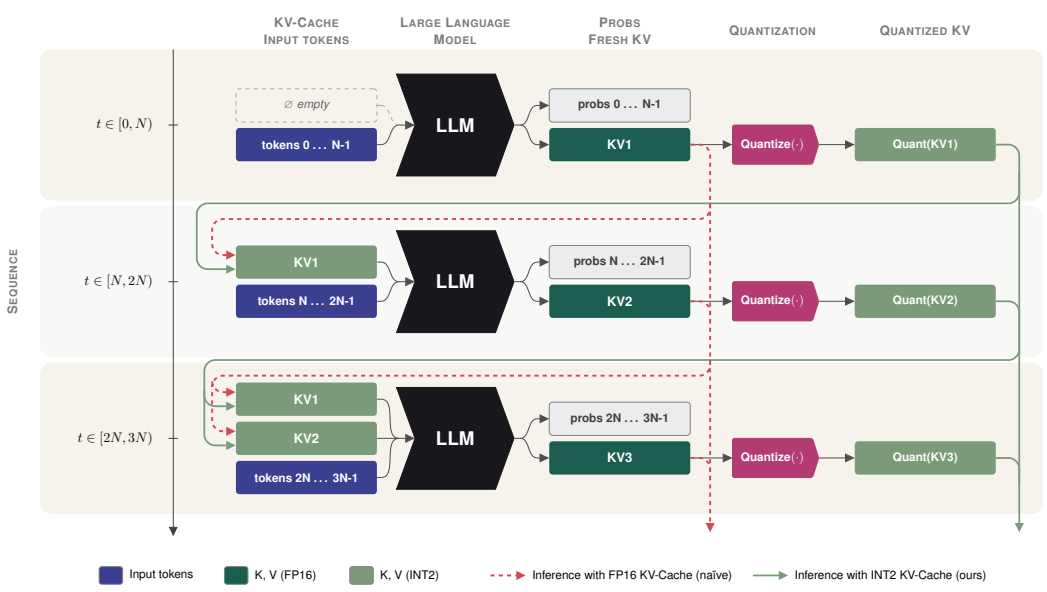


Figure 4: Prior KV-Cache quantization papers address the parallel prefill scenario (red dashed arrows). We propose a 'pseudo-decode' setting (green solid arrows) to better model decoding errors. We split the sequence into blocks of size b. After every block, the freshly produced K, V are quantized before being written back to the KV-cache. Subsequent blocks access a *quantized* cache, so quantization error accumulates over time. LLMs operate this way during decoding of long sequences (e.g., in test-time scaling, such as reasoning tasks). KVarN is designed to operate in this regime.

K and V these are the channel and token dimensions. Prior KV-Cache quantization methods scale only one dimension. In contrast to the Hadamard rotation, this element-wise scaling increases the dequantization overhead only by one FLOP per token per channel. The additional scaling enables the variance normalization below, which we denote by VarN() in the rest of the paper.

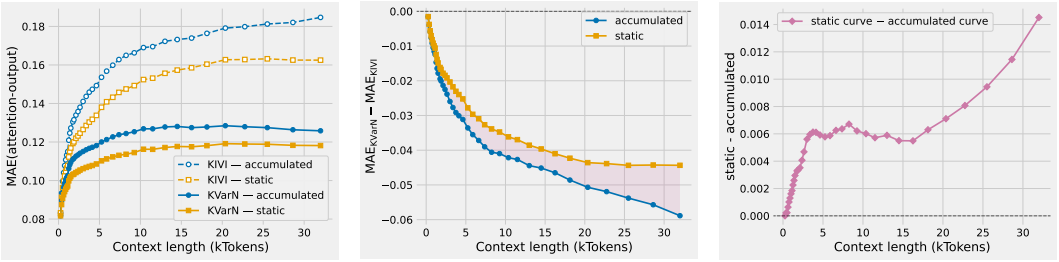
To obtain the two scaling vectors, we iteratively normalize the column-wise and row-wise variance (similar to the modified Sinkhorn normalization of Müller et al. (2025)). Token-axis scaling alone degrades the distribution that round-to-nearest must quantize: applied after the rotation, it *raises* the per-channel excess kurtosis from -0.032 to -0.028 (App. G, Tab. 4). Alternating the two updates normalizes per-token scales without this degradation and suppresses outliers beyond the rotation alone, converging to -0.046 excess kurtosis within eight iterations. This yields well-behaved (in a kurtosis sense) K and V matrices with normalized rows and columns. Before quantization, the variance of the token and channel dimensions of these normalized matrices is uniform.

The two transforms close complementary failure modes and differ in when they act: variance normalization reduces error at all context lengths but, without the rotation, still accumulates over long horizons; the rotation initially amplifies error at 2-bit precision yet suppresses this accumulation. Removing either transform reintroduces its failure mode (App. D, Fig. 10).

3.4 Local Proxy Test: Attention Output Reconstruction

To illustrate the impact of our method, we consider the norm of the reconstruction error for the output of the attention layer under quantized K and V. We evaluate this both under a prefill-like condition that does not consider error-accumulation over quantization steps, and the pseudo-decode setting as given in Sec. 3.2.

In Fig. 5 we see that compared to RTN (KIVI), the proposed KVarN achieves much lower error, and accumulates less error over time. KVarN has much lower token scale errors, see Fig. 2b; in Appendix Fig. 10 we show that specifically the Hadamard rotation mitigates long-term error compounding, while the variance-normalization decreases errors immediately everywhere (and the ideal case being both combined).



(a) Average attention output reconstruction error

(b) Difference of our method to KIVI

(c) Difference between the two curves in (b)

Figure 5: Average reconstruction error after quantization across all attention layer outputs in Qwen3-4B. (a) Our method has smaller error than KIVI at all context lengths. As discussed in Sec. 3.2 errors accumulate over time. Prior work considers the static case (orange) here we consider the dynamic accumulation (blue) (b), (c) KVarN has an increasing advantage over KIVI as contexts get longer.

4 Experiments

4.1 Quantization Quality

In this section, we consider various metrics for the quality of a KV-cache quantization. We focus here on metrics that reveal error-accumulation effects, see Section 3.2.

We evaluate on Qwen3-4B (Yang et al., 2025), Llama-3.1-8B (Grattafiori et al., 2024) and Phi-4-14B (the reasoning-plus variant for AIME, MATH-500 and HumanEval, see App. H) (Abdin et al., 2024) to cover a range of model sizes and families. The Qwen model natively supports reasoning, and for the Phi model, there is a reasoning variant. Llama-3.1-8B does not have a reasoning variant. We evaluate all models on line-retrieval and instruction following and the reasoning models on the reasoning benchmarks. Additional experimental details are discussed in Appendix Sec. H.

Results on the synthetic line-retrieval task, which we find more informative than the commonly used Needle-in-a-Haystack task, are given in Appendix Sec. K.

4.1.1 End-to-End Reasoning and Instruction-Following Performance

The target of KVarN are long-context generation tasks like reasoning, coding and instruction following. Specifically, we consider MATH500 (Lightman et al., 2024), AIME24 (Zhang & Math-AI, 2024), HumanEval (Chen et al., 2021) and IF-Eval (Zhou et al., 2023).

KVarN achieves the best overall performance with the lowest average bits (2.3 per element of KV-Cache), see Tabs. 1, 2, 3.

4.2 Runtime Overhead

In KVarN whenever a new chunk of KV-Cache is stored (e.g., every 128 tokens), it needs to be normalized with iterative variance-scaling. This raises the question: Does this online rescaling operation cause a significant overhead?

To answer this question, we compare the time taken to generate 128 tokens to the time required for 8 iterations of variance-normalization in every attention layer in Qwen3-4B. The base-model is run through vLLM (Kwon et al., 2023) at fp16. On the same hardware (a GPU with 500 TFLOP at fp16 and 1.8 TB/s memory bandwidth) the optimized vLLM token generation takes 1050 ms, while the proposed normalization takes 1.9 ms, see Appendix Fig. 15. This is a 0.18% measured overhead compared to standard methods like KIVI (Liu et al., 2024b) for the quantization. For larger models the relative overhead decreases.

The dequantization operation with two scales instead of one as used in KVarN is known to cause a minor slow-down of about 1% compared to RTN (Müller et al., 2025) (see Appendix Sec. O).

Table 1: Performance on AIME24 and MATH500. Values are Accuracy / # Tokens (mean ± std. err.). Higher is better; best results per column are highlighted. K/V denotes bits per key/value; bits/elem includes auxiliaries and is rounded to one decimal. UP (Uniform Precision) indicates whether all elements within a given K or V tensor share the same precision (✓) or use mixed precision (×).

					AIME24		MATH500	
	Method	**K/V**	**bits/elem**	**UP**	**Acc. (%)**	**# Tokens**	**Acc. (%)**	**# Tokens**
Qwen3-4B	*FP16*	16/16	16.0	✓	*61.1*±1.8	12 477±243	*82.6*±0.3	3857±13
	KIVI	2/2	2.3	✓	55.5±4.0	12 794±188	77.8±0.3	3957±4
	QuaRot	2/2	2.3	✓	56.7±1.9	12 732±112	78.9±0.1	3907±14
	KVQuant-1%	2/2	2.4	×	40.0±1.9	14 794±131	67.5±0.9	5021±26
	PolarQuant	4/2	3.3	✓	52.2±3.3	13 578±208	71.1±0.9	5259±35
	TurboQuant[1]	3/3	4.6	✓	48.9±1.1	13 642±151	77.0±0.5	3917±9
	Kitty	2/2	2.4	×	53.3±5.1	13 123±111	78.5±0.5	3834±23
	KVarN (ours)	2/2	2.3	✓	**60.0**±0.6	12 408±193	**79.2**±0.2	3925±22
Phi-4-14B	*FP16*	16/16	16.0	✓	*62.2*±0.9	11 747±200	*84.9*±0.5	2839±20
	KIVI	2/2	2.3	✓	57.8±1.1	12 529±283	74.4±0.5	3180±16
	QuaRot	2/2	2.3	✓	58.9±1.1	12 284±256	77.0±0.4	3029±54
	KVQuant-1%	2/2	2.4	×	55.6±2.9	13 479±174	72.3±2.9	3588±21
	PolarQuant	4/2	3.3	✓	60.0±1.1	12 347±30	75.8±0.6	3023±12
	TurboQuant[1]	3/3	4.5	✓	52.2±2.9	13 059±129	74.2±0.1	3250±32
	Kitty	2/2	2.4	×	55.6±6.8	12 226±148	74.5±0.3	3171±43
	KVarN (ours)	2/2	2.3	✓	**61.7**±1.0	11 598±136	**84.8**±0.4	2984±39

[1] Results obtained using the vLLM implementation.

Table 2: Performance on HumanEval. Values are Accuracy / # Tokens (mean ± std. err.). Higher accuracy is better; best results per column are highlighted. K/V denotes bits per key/value. UP (Uniform Precision) indicates whether all elements within a given K or V tensor share the same precision (✓) or use mixed precision (×).

				Qwen3-4B		**Phi-4-14B**	
Method	**K/V**	**bits/elem**	**UP**	**Accuracy (%)**	**# Tokens**	**Accuracy (%)**	**# Tokens**
FP16	16/16	16.0	✓	*88.8*±1.0	2764±162	*88.9*±0.6	3015±30
KIVI	2/2	2.3	✓	86.4±0.8	3292±25	74.6±1.4	4929±122
QuaRot	2/2	2.3	✓	86.3±1.1	3255±51	87.0±1.4	3216±5
KVQuant-1%	2/2	2.4	×	85.2±0.4	4594±35	85.6±0.8	4017±70
PolarQuant	4/2	3.3	✓	80.3±1.4	3811±170	86.8±0.2	3378±77
TurboQuant[1]	3/3	4.6	✓	86.2±1.1	3269±83	88.0±0.8	3569±44
Kitty	2/2	2.4	×	86.8±0.5	3085±94	82.7±0.4	3697±85
KVarN (ours)	2/2	2.3	✓	**88.4**±0.2	3192±48	**88.2**±0.2	3220±43

[1] Results obtained using the vLLM implementation.

Most prior methods have larger overhead during dequantization, e.g. for code-book look-ups or for handling mixed-precision.

4.3 End-to-end Throughput

Figure 6 compares AIME accuracy and decode throughput in vLLM across three models. This is a true end-to-end measurement against a practical deployment baseline and includes all operations required to run KVarN. At matched KV-cache precision, KVarN is faster and at least as accurate as TurboQuant, reaching 2× its throughput at 3 bits. At 2 bits (effective total 2.3 bits), KVarN uses ca. 6.9× less KV-Cache memory and is 10-28% faster than the 16-bit baseline with a small accuracy loss, while remaining more accurate than 3-bit TurboQuant. This means that for KVarN the dequantization overhead is more than amortized by the speed gain from reduced data traffic. Measurement details and numerical throughput results are provided in Appendix P.

Table 3: Performance on IFEval. Values are Prompt-Level Strict and Loose accuracy (%). Higher is better; best quantized results per column are bold. K/V denotes bits per key/value. UP (Uniform Precision) indicates whether all elements within a given K or V tensor share the same precision (✓) or use mixed precision (×). bpe indicates the bits per element.

Method	K/V	bpe	UP	Model					
				Qwen3-4B		Llama-3.1-8B		Phi-4-14B	
				Strict	Loose	Strict	Loose	Strict	Loose
FP16	16/16	16.0	✓	81.2±0.2%	86.1±0.4%	72.8±0.3%	81.4±0.4%	63.8±0.3%	70.2±0.1%
KIVI	2/2	2.3	✓	79.4±0.1%	85.4±0.6%	72.2±0.4%	80.3±0.3%	60.9±0.4%	68.1±0.2%
QuaRot	2/2	2.3	✓	79.1±0.4%	85.3±0.5%	70.7±0.5%	79.6±0.3%	62.6±0.3%	68.2±0.1%
KVQuant-1%	2/2	2.4	×	74.9±0.5%	81.9±0.2%	57.7±0.3%	61.4±0.3%	61.4±0.3%	61.8±0.5%
PolarQuant	4/2	3.3	✓	79.2±0.3%	85.4±0.2%	71.2±0.9%	79.6±0.6%	62.3±0.2%	68.3±0.1%
TurboQuant[1]	3/3	4.6	✓	79.9±0.1%	85.7±0.1%	66.9±0.5%	76.0±0.1%	59.2±0.5%	69.3±0.3%
Kitty	2/2	2.4	×	79.2±0.1%	85.3±0.2%	70.7±0.5%	79.3±0.4%	63.2±0.2%	68.4±0.3%
KVarN (ours)	2/2	2.3	✓	**80.6**±0.4%	**85.9**±0.2%	**72.4**±0.1%	**80.9**±0.2%	**63.7**±0.1%	**69.9**±0.2%

[1] Results obtained using the vLLM implementation.

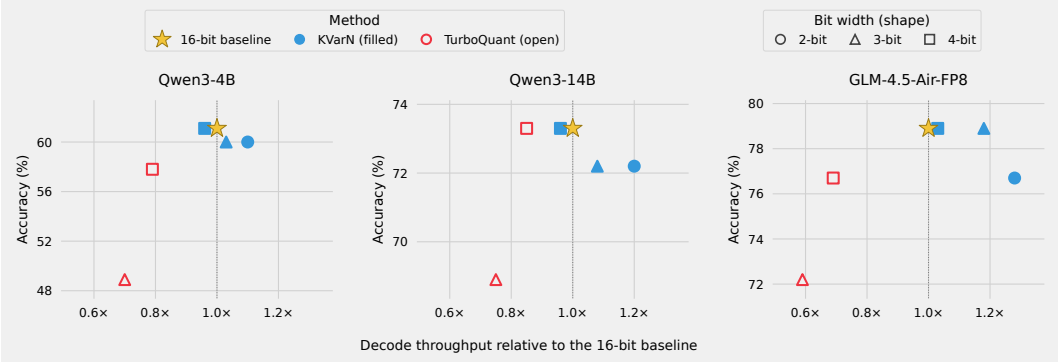


Figure 6: AIME accuracy versus decode throughput on Qwen3-4B, Qwen3-14B, and GLM-4.5-Air-FP8, normalized to each model's 16-bit KV-cache baseline (yellow star). Upper right is better. Filled blue and open red markers denote KVarN and TurboQuant; circles, triangles, and squares indicate 2, 3, and 4 bits. KVarN at 3 bits uses 4-bit keys and 2-bit values. Accuracy scales differ across panels. GLM-4.5-Air-FP8 uses tensor parallelism over two GPUs (with 500 TFLOP at fp16 and 1.8TB/s memory bandwidth).

5 Related Work

5.1 KV-Cache Quantization

Most closely related to our approach are works on KV cache quantization, which aim to reduce memory usage by converting full-precision keys and values to low-bit representations while preserving model performance. Early methods such as KIVI (Liu et al., 2024b) observe that keys and values exhibit distinct statistical properties, and therefore propose treating them differently, quantizing keys per channel and values per token to better capture their variability. Building on this idea, KVQuant (Hooper et al., 2024) introduces non-uniform quantization combined with outlier-aware handling, retaining a small fraction of high-magnitude channels (around 1%) in full precision to maintain accuracy. Other approaches like Kitty (Xia et al., 2025) proposes a 2-bit quantization scheme augmented with channel-wise importance selection, where a subset of key channels (e.g., the top 12%) is stored at higher precision (4-bit). OSCAR (Zhou et al., 2026) derives rotations for KV-cache quantization from offline, per-model covariance profiling; in contrast, KVarN is calibration-free and requires no offline profiling stage.

In a different direction, PolarQuant (Wu et al., 2025) represents keys in polar coordinates, quantizing radius and angle separately while applying more aggressive compression to values. Recently, TurboQuant and NSNQuant (Zandieh et al., 2026; Son et al., 2025) frame KV cache compression as a vector quantization problem. Both apply lightweight preprocessing to ease the cache quantization. TurboQuant further refines the process through a lightweight residual correction based on a 1-bit quantized Johnson–Lindenstrauss transform to better preserve inner products. NSNQuant applies a fixed, non-iterative normalization before codebook vector quantization. KVarN differs in two ways: (i) iterative alternating variance-balancing retains the token-scale benefit without the kurtosis degradation that direct scaling causes and (ii) KVarN targets linear RTN, keeping dequantization a fused single-pass kernel rather than a codebook lookup — which is what makes exceeding FP16 throughput possible (Sec. 4.3); see also App. G for further details. Our approach builds primarily on KIVI (Liu et al., 2024b), and through our analysis find targeted improvements to form KVarN.

5.2 KV-Cache Compression with Eviction and/or Token-Merging

KV cache compression can also be achieved by deleting or merging tokens. Eviction prunes token-level cache entries. H2O (Zhang et al., 2023) and Scissorhands (Liu et al., 2023) estimate token importance from attention scores, retaining entries that receive high attention from subsequent queries. SnapKV (Li et al., 2024) and PyramidKV (Cai et al., 2025) use queries from a trailing context window to score KV pairs, incorporating future information into eviction. KVZip (Kim et al., 2025) instead takes a query-agnostic approach, scoring entries by their contribution to reconstructing the original context through teacher-forced decoding. Complementing eviction, token merging combines tokens into a smaller set of representatives. CaM (Zhang et al., 2024) merges value states via Bernoulli masking, uniformly across layers; D2O (Wan et al., 2025) instead allocates cache capacity per layer and adapts merging with an exponential-moving-average threshold. Token-merging and eviction are largely orthogonal to quantization and can be combined with it. However, for completeness we add comparisons to such methods in the Appendix, see Sec. J.

5.3 Preconditioning in Weight Quantization

The distribution of weight tensors can often be made more favorable by applying some simple transformations to them (Ashkboos et al., 2024; Liu et al., 2025). Particularly useful are transformations that can be either applied cheaply on-the-fly or precomputed and absorbed into existing model weights. The Hadamard transform is fast enough for online application (with $O(N \log N)$ complexity) and can often be absorbed into adjacent weight matrices (see App. Fig. 7). We find that in KV-Cache quantization, incoherence processing is helpful to equalize channel-space outliers and to mitigate error accumulation when applied to pre-normalized tokens, but insufficient on its own to manage token-wise scaling errors, see Fig. 2. An alternative preconditioning transform has recently become popular in LLM weight quantization: Scaling both input- and output-channel dimensions to have approximately uniform variance by way of a variance-targeted Sinkhorn-Knopp-style normalization. This normalization is helpful because it approximates calibration data from weight structure. Thereby, it improves end-to-end output quality while it paradoxically causes weight-matrix reconstruction errors to increase (Müller et al., 2025). We find that surprisingly, variance normalization is helpful in KV-Cache quantization, but for unrelated reasons as there is no calibration data to approximate; here instead it helps mitigate the damage caused by impactful token-scale errors, see Fig. 1.

6 Conclusion

In this paper we have shown that under current KV-Cache quantization methods, end-to-end output degradation is driven by outlier errors that are brought about by incorrect token scaling. We further show that quantization error accumulates across decoding timesteps as later tokens repeatedly attend to an already-quantized cache, an effect that the static prefill evaluations used by existing methods cannot observe, and propose a pseudo-decode diagnostic that makes this accumulation measurable. We address token scaling with KVarN, which also suppresses the error accumulation that arises in long decoding tasks under KV-Cache quantization. With KVarN we achieve state-of-the-art quality on generative benchmarks including AIME24, MATH-500, HumanEval and IFEval, enabling near-lossless 2.3-bit-per-element KV-Caches (a $6.9\times$ reduction from standard 16-bit KV-Cache) and up to $2\times$ end-to-end speedup over the strong TurboQuant vLLM baseline.

AI USE STATEMENT

In this work, we used generative AI tools to assist in all code creation including for the core quantization methods and triton kernels. We have not used generative AI tools for any of these:

- help develop theoretical models or conceptual frameworks,
- formulate mathematical claims,
- propose or refine hypotheses,
- design or provide feedback on research methodology or experiments,
- support qualitative and thematic data analysis,
- interpret results.

The following possible uses of generative AI tools do not apply to this paper:

- assist with translation
- assist in the writing of proofs,
- clean and reformat datasets
- generate synthetic data sets,
- provide critical ingredients for proving mathematical claims.

Furthermore generative AI tools were used to polish text. We take responsibility for the final content, claims, and artifacts in and associated with our paper.

ETHICS STATEMENT

This work concerns the efficiency of LLM inference: it makes processes that are already fully feasible faster and cheaper, and does not fundamentally alter their societal impact. All experiments use openly available models and standard public benchmarks; no human subjects or personal data are involved. We have read and adhere to the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

Code implementing KVarN and reproducing the key experiments is provided in the supplementary material. All models and datasets used are openly available. Algorithmic details of the variance normalization are given in Alg. 1 and Appendix Q reports the compute required to reproduce the results; further experimental details are listed in the appendix.

REFERENCES

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

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

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. 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.

Zhenni Bi, Kai Han, Chuanjian Liu, Yehui Tang, and Yunhe Wang. Forest-of-thought: Scaling test-time compute for enhancing LLM reasoning. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 4253–4267. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/bi25a.html.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=ayi7qezU87.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

Greg Kamradt. Needle in a haystack-pressure testing llms. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2023.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. KVzip: Query-agnostic KV cache compression with context reconstruction. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=JFygzwx8SJ.

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 *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. How long can context length of open-source LLMs truly promise? In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*, 2023. URL https://openreview.net/forum?id=LywifFNXV5.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 22947–22970. Curran Associates, Inc., 2024. doi: 10.52202/079017-0722. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/28ab418242603e0f7323e54185d19bde-Paper-Conference.pdf.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024a.

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 *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ogO6DGE6FZ.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 52342–52364. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a452a7c6c463e4ae8fbdc614c6e983e6-Paper-Conference.pdf.

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. *Proceedings of Machine Learning Research*, 235:32332–32344, 2024b.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321, 2025.

Lorenz K Müller, Philippe Bich, Jiawei Zhuang, Ahmet Çelik, Luca Benfenati, and Lukas Cavigelli. Sinq: Sinkhorn-normalized quantization for calibration-free low-precision llm weights. *arXiv preprint arXiv:2509.22944*, 2025.

OpenAI. Learning to reason with llms, September 2024. URL https://openai.com/index/learning-to-reason-with-llms/. Accessed: May 2026.

Junhong Shen, Hao Bai, Lunjun Zhang, Yifei Zhou, Amrith Setlur, Shengbang Tong, Diego Caples, Nan Jiang, Tong Zhang, Ameet Talwalkar, and Aviral Kumar. Thinking vs. doing: Improving agent reasoning by scaling test-time interaction. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=un1TRwNgiv.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Shriyank Somvanshi, Md Monzurul Islam, Mahmuda Sultana Mimi, Sazzad Bin Bashar Polock, Gaurab Chhetri, and Subasish Das. From s4 to mamba: A comprehensive survey on structured state space models. *arXiv preprint arXiv:2503.18970*, 2025.

Donghyun Son, Euntae Choi, and Sungjoo Yoo. NSNQuant: A double normalization approach for calibration-free low-bit vector quantization of KV cache. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=boNYskaXnO.

Alon Talmor and Jonathan Berant. MultiQA: An empirical investigation of generalization and transfer in reading comprehension. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4911–4921, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1485. URL https://aclanthology.org/P19-1485/.

Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, Longyue Wang, and Mi Zhang. D2O: Dynamic discriminative operations for efficient long-context inference of large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=HzBfoUdjHt.

Songhao Wu, Ang Lv, xiao feng, Yufei zhang, Xun Zhang, Guojun Yin, Wei Lin, and Rui Yan. Polarquant: Leveraging polar transformation for key cache quantization and decoding acceleration. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=JCTTLKEBza.

Haojun Xia, Xiaoxia Wu, Jisen Li, Robert Wu, Junxiong Wang, Jue Wang, Chenxi Li, Aman Singhal, Alay Dilipbhai Shah, Alpay Ariyak, et al. Kitty: Accurate and efficient 2-bit kv cache quantization with dynamic channel-wise precision boost. *arXiv preprint arXiv:2511.18643*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. Turboquant: Online vector quantization with near-optimal distortion rate. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tO3ASKZlok.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2024, 2024.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. Cam: Cache merging for memory-efficient LLMs inference. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=LCTmppB165.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b15572587b78ecfcebb2827f8-Paper-Conference.pdf.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

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. *arXiv preprint arXiv:2605.17757*, 2026.

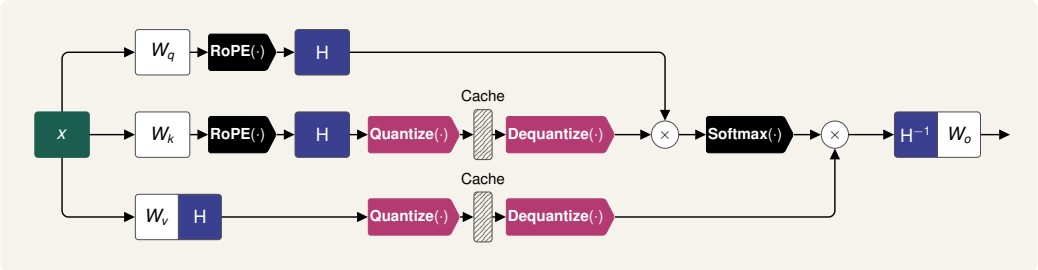


Figure 7: Arrangement of Hadamard transforms in an attention layer in our method.

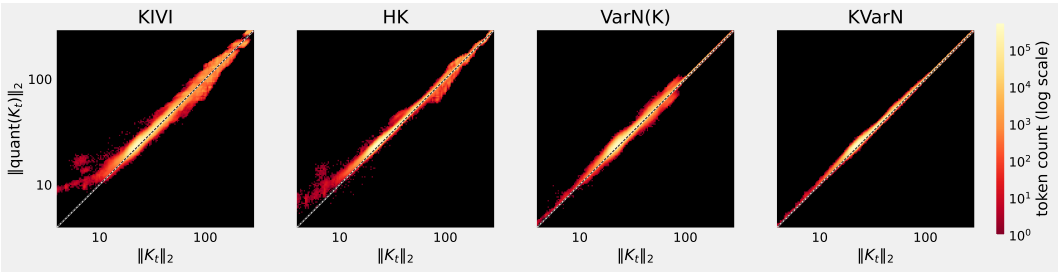


Figure 8: Joint distribution of K magnitude and quantized K magnitude using different quantization methods. KVarN tightly controls token scales, while the baseline KIVI and ablations of our method have substantial off-diagonals.

A Hadamard Transform Arrangement

The arrangement of Hadamard transforms we use is similar to QuaRot (Ashkboos et al., 2024), see Fig. 7. There are two absorbed transforms (that are merged with W_V and W_O respectively and do not need to be computed during inference) and two online transforms after the RoPE-embedding. The two online transforms are head-wise, i.e. seen on the full layer they are block-diagonal Hadamard transforms with a block-size equal to the head-size. The complexity of these online transforms is $O(N \log N)$, which is negligible compared to the linear layers next to them (Ashkboos et al., 2024).

B Full Token-Scale Distribution Error

We provide the full empirical joint distribution of per token K magnitude before and after quantization in Fig. 8 using different ablations of our method. KVarN successfully contains the magnitudes to the diagonal. We see that Hadamard rotation and variance-normalization have different, complementary effects: Hadamard rotation squeezes the distribution close to the identity, but at very large and very small tokens performs much worse than variance normalization. Variance Normalization in contrast is especially effective at these extremes and in KVarN we clearly see the synergistic effect of both in KVarN.

C Magnitude Error Quantiles by Dataset

In Fig. 9 we show that the magnitude error quantiles of Fig. 2 look similar for different datasets (wikitext2 and c4).

D Which Parts of KVarN Mitigate Error Accumulation?

In Fig. 10 we show an ablation of Fig 5c. We find that it is crucial to add the Hadamard transform onto VarN to mitigate the error accumulation effect over very long contexts (possibly due to channel

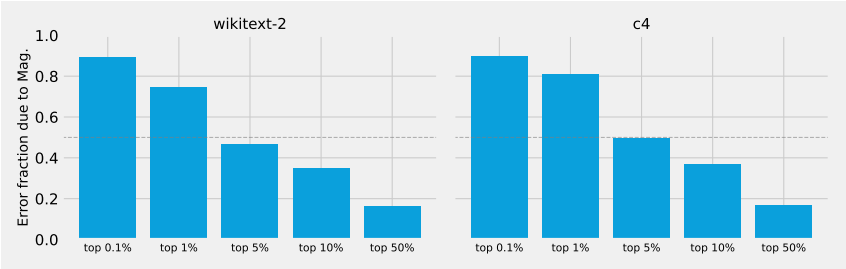


Figure 9: Error fraction in K tensor due to magnitude errors for different datasets. Compare Fig. 2.

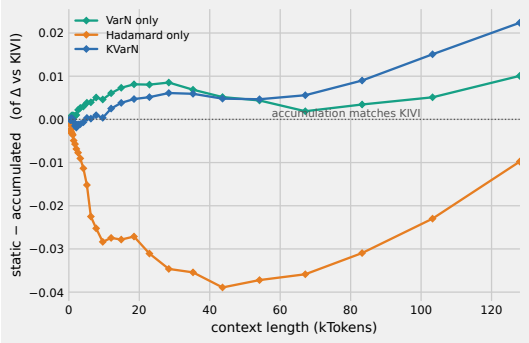


Figure 10: Ablation of Fig. 5c. Both variance-normalization and Hadamard transform are needed for KVarN's stable long horizon behaviour.

decorrelation). On its own Hadamard is insufficient for this because it initially causes much larger errors. Conversely, variance-normalization decreases errors immediately, but here the errors accumulate similarly to KIVI over very long horizons. KVarN combines the two beneficial effects.

E Does KVarN Improve Quality at FP8?

To show that KVarN improves KV-Cache quality also at FP8, we include here Fig. 11 analogous to Fig. 5b. Also at FP8 KVarN can be used to get quality improvements. At FP8 the dominant contribution stems from the Hadamard rotation (in contrast to the 2-bit regime, Fig. 12).

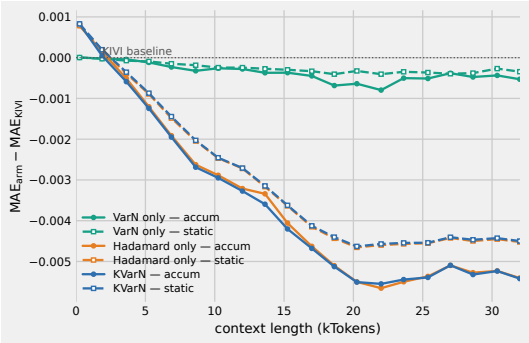


Figure 11: Also at FP8 KVarN can improve quantization fidelity (lower is better; < 0 implies better quality than KIVI).

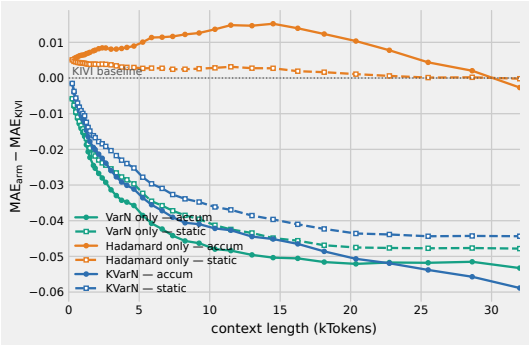


Figure 12: At 2-bit KV cache, hadamard alone does not improve over KIVI (c.f. Fig. 11), but combined with variance normalization helps mitigate error accumulation.

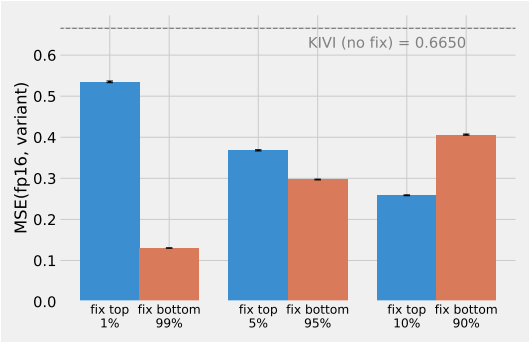


Figure 13: Plot to complement Fig. 1a. It shows much MSE remains, if we replace the largest top k% errors with the high precision value. Note that the top 5% of entries contribute less to MSE than the bottom 95%, but their end-to-end KL-divergence impact is greater than that of those bottom 95%.

F Outlier Contributions to MSE

In Fig. 13 we show which error quantiles contribute how much to the total MSE of the model. This is best compared with Fig. 1a, where we show the same quantile contributions for end-to-end KL-divergence. Crucially, the 5% largest errors cause a minority of the MSE, but a majority of the end-to-end KL-divergence. In this sense we can say that fixing outliers is disproportionally important.

G Kurtosis Evolution Across Iterations

To examine how iterative balancing interacts with outlier suppression, we measure the block-wise excess kurtosis of the projection matrices at each transformation stage. As shown in Table 4, the Hadamard rotation substantially suppresses outliers, reducing kurtosis from 0.115 to -0.032. Adding token scaling without iterative rebalancing partially offsets this improvement, increasing kurtosis to -0.028.

Alternating balancing steps preserve the benefits of token-magnitude scaling while further suppressing outliers. Kurtosis decreases consistently over the first several iterations and stabilizes near -0.047 after about eight iterations.

Note that direct token scaling is used in other methods, e.g., NSNQuant (Son et al., 2025). Direct token scaling damages the block-wise excess kurtosis; NSNQuant presumably absorbs this with a codebook matched to the resulting distribution, at the cost of codebook lookups during dequantization (cf. TurboQuant, Sec. 4.3). For fast dequantization, as KVarN demonstrates with linear quantization, iterative rebalancing becomes necessary to maintain quality.

Table 4: Block-wise excess kurtosis of the projection matrices under different rotation and scaling configurations. Iterative Sinkhorn balancing progressively reduces kurtosis, counteracting the increase introduced by non-iterative input-channel scaling.

Setting	Kurtosis
Raw (no Hadamard)	0.115
Hadamard only	−0.032
Hadamard + direct token scaling	−0.028
Hadamard + Sinkhorn, iterations = 1	−0.038
Hadamard + Sinkhorn, iterations = 2	−0.042
Hadamard + Sinkhorn, iterations = 4	−0.044
Hadamard + Sinkhorn, iterations = 8	−0.046
Hadamard + Sinkhorn, iterations = 16	**−0.047**
Hadamard + Sinkhorn, iterations = 32	−0.046

H EXPERIMENTAL DETAILS

Models. We evaluate four models spanning different capability profiles and parameter scales: Qwen3-4B (`Qwen/Qwen3-4B`), Llama-3.1-8B-Instruct (`meta-llama/Llama-3.1-8B-Instruct`), Phi-4 (`microsoft/phi-4`), Phi-4-reasoning-plus (`microsoft/Phi-4-reasoning-plus`), and Qwen3-14B (`Qwen/Qwen3-14B`). Qwen3-4B is evaluated on all tasks (with reasoning mode activated only for AIME 2024, MATH500 and HumanEval). Phi-4 is used for IFEval and Line Retrieval only and Phi-4-reasoning-plus for AIME 2024, MATH-500, and HumanEval.

Quantization. All quantized runs use 2-bit KV cache compression with a fixed three-region layout: S sink tokens in FP16 (preserving attention sink behavior), a 2-bit quantized body composed of groups of G tokens each, and R trailing tokens in FP16 corresponding to the most recently generated tokens, which have not yet accumulated enough context to form a full quantization group. We use a classical setting with G=128, S=128, R=128 throughout, except for IFEval where S=32. Methods that prescribe mixed-precision within groups, such as KVQuant (Hooper et al., 2024), PolarQuant (Wu et al., 2025), and TurboQuant (Zandieh et al., 2026), retain their original per-element precision allocation inside each group. In KVarN, zeropoints are stored in FP16 and scales in FP8.

Decoding. We follow the recommended decoding parameters for each model. Qwen3-4B uses thinking mode for AIME 2024, MATH-500 and HumanEval (temp. 0.6, top-p 0.95, top-k 20) and non-thinking mode elsewhere (temp. 0.7, top-p 0.8, top-k 20). Llama-3.1-8B-Instruct uses temp. 0.6, top-p 0.9. Phi-4 decodes greedily (variance between runs is non-zero due to floating point arithmetic ordering sensitivity and non-deterministic ordering on GPU). Phi-4-reasoning-plus uses temp. 0.8, top-p 0.95, top-k 50 with the official model-card system prompt.

IFEval. Full `google/IFEval` benchmark (Zhou et al., 2023) (541 prompts, `max_new_tokens=1280`).

MATH-500 and AIME 2024. MATH-500 (Lightman et al., 2024) (500 problems, `max_new_tokens=8192`; for Fig. 1c `max_new_tokens=16384`); answers extracted from `\boxed{}`. AIME 2024 (Zhang & Math-AI, 2024) uses the 30 competition problems from `AI-MO/aimo-validation-aime`, with `max_new_tokens=16384` to accommodate extended chain-of-thought reasoning. Both benchmarks report Avg@3, averaged over 3 independent runs.

HumanEval Extended version of HumanEval (Chen et al., 2021) (164 problems, `max_new_tokens=16384`), graded by execution with a 30 s timeout. We report Avg@3.

Line Retrieval. Following Li et al. (2023), we construct contexts of $L \in \{100, 200, 300, 400, 500, 600\}$ numbered lines, each holding a random 10-character alphanu-

meric code drawn uniformly from $\{\mathrm{A}, \ldots, \mathrm{Z}\} \cup \{0, \ldots, 9\}$. The model is asked to retrieve the code at a randomly chosen line. We use 100 samples per L (`max_new_tokens`$= 32$) and report exact-match accuracy. The prompt follows the template:

```
Below is a list of numbered lines, each with a unique
10-character alphanumeric code.
line 1:  <c1>
line 2:  <c2>
...
What is the code on line k?  Reply with only the
10-character code, nothing else.
```

The predicted answer is the last 10-character alphanumeric token found in the output.

Needle-in-a-Haystack (NIAH). We use the benchmark of Kamradt (2023) with Paul Graham essays as the haystack. The needle is inserted at depth $d \in \{0.1, \ldots, 0.9, 1\}$, where d denotes the fractional position within the haystack, across context lengths spanning 10-100% of Qwen3-4B's 32 768-token window (3112 to 31 129 tokens, yielding a 10×10 evaluation grid (`max_new_tokens=128`). The *Static* setting uses the standard quantization protocol, while *Accumulated* uses the pseudo-decode setting introduced in Figure 4. The prompt follows the template:

> *You are a helpful assistant. Read the following long passage carefully and remember its contents. I will quiz you about it at the end.*
>
> *<haystack with needle inserted at depth d>*
>
> *What is the best thing to do in San Francisco?*

where the needle is the verbatim sentence: *"The best thing to do in San Francisco is eat a sandwich and sit in Dolores Park on a sunny day"* (Kamradt, 2023). A response scores 10 if both *sandwich* and *Dolores* appear (case-insensitive), 5 if exactly one, and 0 otherwise.

Baselines. KVQuant (Hooper et al., 2024) additionally retains 1% of K and V channels in FP16. PolarQuant (Wu et al., 2025) requires 4-bit precision for the key cache to yield meaningful results. TurboQuant (Zandieh et al., 2026) is evaluated in a 3-bit K / 3-bit V configuration; as no official implementation is available, we use the community implementation merged into the vLLM framework (Kwon et al., 2023)[3], in which the KV cache of the first and last two layers is left unquantized.

Effective memory overhead. All bits-per-element figures reported in the tables include auxiliary storage (scales and zero-points) in addition to the quantized values. For our method, K and V elements are stored at 2 bits with two FP8 scales (one of which absorbs the Sinkhorn normalisation scale into the RTN scale) and one FP16 zero-point per group of $G{=}128$ elements, giving 2.25 bits/element (in tables rounded to 2.3 for legibility). For KVQuant (Hooper et al., 2024), the original configuration applies 2-bit group quantization ($G{=}128$) with FP16 scale and zero-point, plus the top-1% of channels retained in FP16, yielding ≈ 2.4 bits/element. For TurboQuant (Zandieh et al., 2026), the publicly available implementation leaves the KV cache of the first and last two transformer layers unquantized (FP16), making the effective overhead model-dependent; when a single figure is reported it is the average across all layers for that model.

I LIMITATIONS

Some novel LLM architectures do not require a KV-Cache (e.g. state-space models SSMs (Somvanshi et al., 2025)). Our method is not suitable for such architectures.

Some recent models use train-time compression for the KV-Cache in the form of multi-head latent attention (MLA) (Liu et al., 2024a). As in other KV-Cache compression methods, it is unclear how such attention mechanisms affect quantization quality.

Some performance margins fall within run-to-run noise.

Our diagnostic analysis uses one model under one quantizer.

[3] `https://github.com/vllm-project/vllm/pull/38479`

Table 5: Performance across context lengths (number of lines). Values are accuracy (%). Higher is better; best results per column are highlighted. K/V denotes bits per key/value. The equivalent bits/elem reflects the effective compression ratio. UP (Uniform Precision) indicates whether all elements within a given K or V tensor share the same precision (✓) or use mixed precision (×).

Model	Method	K/V	equivalent bits/elem	UP	Number of lines					
					100	200	300	400	500	600
Llama-3.1-8B	*FP16*	16/16	16.0	✓	*100%*	*99%*	*99%*	*94%*	*93%*	*91%*
	SnapKV (7×)	16/16	2.3	✓	77%	87%	90%	90%	88%	85%
	PyramidKV (7×)	16/16	2.3	✓	61%	88%	87%	88%	88%	87%
	KVZip (7×)	16/16	2.3	✓	80%	92%	86%	88%	89%	86%
	KVarN (ours)	2/2	2.3	✓	**99%**	**98%**	**98%**	**95%**	**91%**	**89%**

J Comparison with Eviction Methods

In Tab. 5, we compare against KV cache eviction approaches such as SnapKV (Li et al., 2024), PyramidKV (Cai et al., 2025), and KVZip (Kim et al., 2025) on the Line Retrieval task with Llama-3.1-8B-Instruct. Unlike quantization, which compresses the KV cache uniformly across both prefill and generation, these methods apply compression only to the prompt cache and leave the generation cache untouched. This makes them orthogonal to quantization and explains why we restrict the comparison to this appendix. To match the effective memory footprint of our method (2.3 bits per element on average), we apply a 7× compression ratio to the prompt cache.

K Line-Retrieval Accuracy

Because synthetic tasks, especially NiaH (Kamradt, 2023), are commonly used in prior work, we add line-retrieval (Li et al., 2023). While similar to NiaH we find line-retrieval to be more informative. NiaH seems comparatively too easy, especially, when error accumulation across time is not taken into account. See Appendix Sec. L for NiaH evaluations. We give comprehensive results on line-retrieval for various baselines and models, see Tab. 6. KVarN performs best overall.

L NiaH: Needle-in-a-Haystack

We also evaluate on the Needle-in-a-Haystack task (Kamradt, 2023). The *Static* setting involves prefill only and thus no error accumulation, making the *Accumulated* pseudo-decode setting (Section 3.2) essential to assess retrieval quality under realistic decoding conditions. Further experimental details are provided in Appendix H. Crucially, the static version of NiaH that is commonly used for KV-Cache compression evaluation is clearly much easier than the accumulated version in pseudo-decode setting.

M Runtime Overhead Figure

In Fig. 15 we see that the variance normalization of KVarN introduces a small runtime overhead that can easily be amortized in memory bound scenarios. This measurement is added to aid comparison to other *quantization* methods that do not use variance normalization, but have similar (KIVI) or worse (codebook-based) overheads at dequantization.

N Variance Normalization Algorithm

We perform the variance normalization on the K and V matrices using Alg. 1, adapted from Müller et al. (2025).

Table 6: Performance across context lengths (number of lines). Values are accuracy (%). Higher is better; best results per column are highlighted. K/V denotes bits per key and value. UP (Uniform Precision) indicates whether all elements within a given K or V tensor share the same precision (✓) or use mixed precision (×).

					Number of lines					
Model	**Method**	**K/V**	**bits/elem**	**UP**	**100**	**200**	**300**	**400**	**500**	**600**
Qwen3-4B	*FP16*	16/16	16.0	✓	*100%*	*99%*	*98%*	*98%*	*96%*	*90%*
	KIVI	2/2	2.3	✓	98%	97%	88%	84%	84%	74%
	Hadamard	2/2	2.3	✓	98%	98%	91%	**98%**	92%	83%
	KVQuant-1%	2/2	2.4	×	93%	80%	82%	69%	57%	57%
	PolarQuant	4/2	3.3	✓	93%	86%	83%	91%	82%	84%
	TurboQuant[1]	3/3	4.6	✓	99%	99%	**98%**	95%	94%	**85%**
	Kitty	2/2	2.4	×	**100%**	99%	93%	94%	95%	79%
	KVarN (ours)	2/2	2.3	✓	**100%**	**99%**	**98%**	97%	**97%**	**85%**
Llama-3.1-8B	*FP16*	16/16	16.0	✓	*100%*	*99%*	*99%*	*96%*	*93%*	*92%*
	KIVI	2/2	2.3	✓	98%	97%	90%	84%	80%	83%
	Hadamard	2/2	2.3	✓	**99%**	97%	94%	90%	84%	85%
	KVQuant-1%	2/2	2.4	×	92%	88%	77%	68%	58%	54%
	PolarQuant	4/2	3.3	✓	59%	72%	76%	77%	67%	58%
	TurboQuant[1]	3/3	4.6	✓	98%	95%	96%	88%	79%	83%
	Kitty	2/2	2.4	×	98%	**98%**	95%	87%	80%	87%
	KVarN (ours)	2/2	2.3	✓	**99%**	**98%**	**98%**	**95%**	**91%**	**89%**
Phi-4-14B	*FP16*	16/16	16.0	✓	*100%*	*100%*	*100%*	*98%*	*100%*	*95%*
	KIVI	2/2	2.3	✓	95%	96%	88%	88%	91%	82%
	Hadamard	2/2	2.3	✓	99%	**99%**	97%	**97%**	91%	94%
	KVQuant-1%	2/2	2.4	×	94%	80%	86%	82%	74%	67%
	PolarQuant	4/2	3.3	✓	98%	**99%**	69%	**97%**	80%	85%
	TurboQuant[1]	3/3	4.5	✓	61%	51%	55%	53%	57%	56%
	Kitty	2/2	2.4	×	99%	98%	97%	93%	96%	83%
	KVarN (ours)	2/2	2.3	✓	**100%**	**99%**	**99%**	**97%**	**98%**	**95%**

[1] Results obtained using the vLLM implementation.

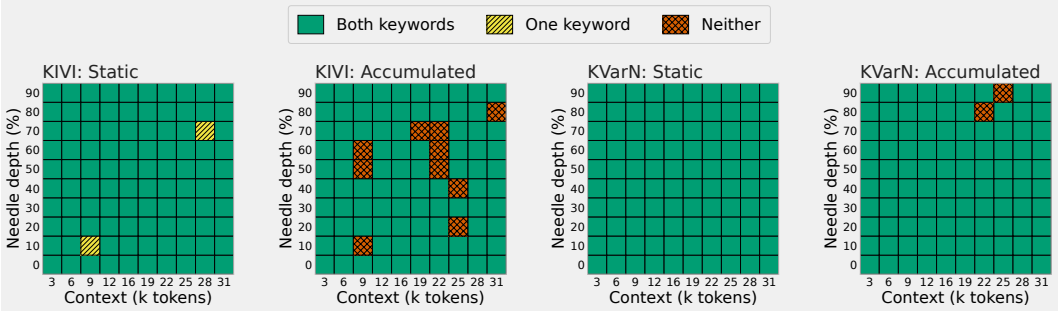


Figure 14: Needle-in-a-Haystack on Qwen3-4B: KIVI vs. KVarN under Static and Accumulated prefill. Cells are colored and hatched by retrieval outcome (both keywords / one keyword / neither).

O KVarN Dequantization Overhead

We measure the wall-clock cost of dequantizing a full layer of the quantized KV cache (16 attention heads, head dimension 128, group size 128) for context lengths 4k-32k tokens. The KIVI baseline performs standard single-scale dequantization. KVarN additionally requires a per-row second scale s_2 from its dual scaling, which we fuse into the dequant kernel so that no extra HBM round-trip is incurred.

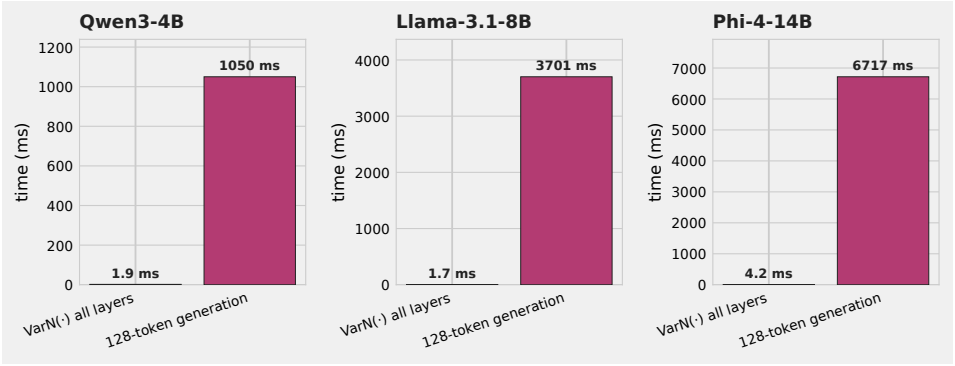


Figure 15: Speed measurement on GPU in the fast vLLM framework. The variance-normalization causes a very minor overhead.

Algorithm 1 VarN, Variance-Normalization by Alternating Log-Domain Dual-Scale Balancing

Require: Batched tiles $\mathbf{T} \in \mathbb{R}^{N\times R\times C}$, Iterations K, Limits $c_{\text{min}}, c_{\text{max}}$
Ensure: Balanced tiles $\mathbf{T}_{\text{bal}}$, Scales $\mathbf{S}_c \in \mathbb{R}^{N\times 1\times C}, \mathbf{S}_r \in \mathbb{R}^{N\times R\times 1}$

function IMB($\mathbf{X}$)
  $\vec{v}_c \leftarrow \text{Var}_{\text{col}}(\mathbf{X})$; $\vec{v}_r \leftarrow \text{Var}_{\text{row}}(\mathbf{X})$ ▷ Variances across dims 2 and 1
  return $\frac{\max(\vec{v}_c)}{\max(\min(\vec{v}_c),10^{-8})} + \frac{\max(\vec{v}_r)}{\max(\min(\vec{v}_r),10^{-8})}$
end function
$\mathbf{L}_c \leftarrow \mathbf{0}_{N\times 1\times C}$; $\mathbf{L}_r \leftarrow \mathbf{0}_{N\times R\times 1}$ ▷ Initialize log-scales
$\mathbf{C} \leftarrow (\mathbf{T} \oslash \exp(\mathbf{L}_c)) \oslash \exp(\mathbf{L}_r)$ ▷ Current normalized tiles
$I_{\text{best}} \leftarrow \text{Imb}(\mathbf{C})$
$\mathbf{S}_c^* \leftarrow \exp(\mathbf{L}_c)$; $\mathbf{S}_r^* \leftarrow \exp(\mathbf{L}_r)$ ▷ Track best linear scales
for $k \leftarrow 1$ to K **do**
  $\mathbf{v}_{\text{col}} \leftarrow \text{clamp}(\text{Var}_{\text{row}}(\mathbf{C}), c_{\text{min}}, c_{\text{max}})$
  $\mathbf{L}_c \leftarrow \text{clamp}(\mathbf{L}_c + 0.5 \cdot \log(\mathbf{v}_{\text{col}}), -0.3, 10.0)$
  $\mathbf{C} \leftarrow (\mathbf{T} \oslash \exp(\mathbf{L}_c)) \oslash \exp(\mathbf{L}_r)$
  $\mathbf{v}_{\text{row}} \leftarrow \text{clamp}(\text{Var}_{\text{col}}(\mathbf{C}), c_{\text{min}}, c_{\text{max}})$
  $\mathbf{L}_r \leftarrow \text{clamp}(\mathbf{L}_r + 0.5 \cdot \log(\mathbf{v}_{\text{row}}), -0.3, 10.0)$
  $\mathbf{C} \leftarrow (\mathbf{T} \oslash \exp(\mathbf{L}_c)) \oslash \exp(\mathbf{L}_r)$
  $I_{\text{curr}} \leftarrow \text{Imb}(\mathbf{C})$
  for $n \leftarrow 1$ to N **do** ▷ Batched conditional update over N tiles
    if $I_{\text{curr}}[n] \leq I_{\text{best}}[n]$ **then**
      $I_{\text{best}}[n] \leftarrow I_{\text{curr}}[n]$
      $\mathbf{S}_c^*[n] \leftarrow \exp(\mathbf{L}_c[n])$; $\mathbf{S}_r^*[n] \leftarrow \exp(\mathbf{L}_r[n])$
    end if
  end for
end for
return $(\mathbf{T} \oslash \mathbf{S}_c^*) \oslash \mathbf{S}_r^*,\ \mathbf{S}_c^*,\ \mathbf{S}_r^*$

Figure 16 reports the median time per call (multiple repeated runs) for both kernels, implemented in Triton on GPU. Across all context lengths the gap between KVarN and KIVI is at most 1.4% and within measurement noise at 16k and 32k tokens. This shows that KVarN's dual scaling adds negligible runtime overhead over the single-scale baseline once s_2 is fused into the dequant kernel.

P END-TO-END DECODE THROUGHPUT

Table 7 reports the numerical results corresponding to Figure 6. We evaluate KVarN and the TurboQuant implementation available in vLLM using 256 distinct concurrent prompts, without prefix reuse, a 12K-token context, and 512 generated tokens per prompt. Throughput is normalized to the corresponding model's 16-bit KV-cache baseline.

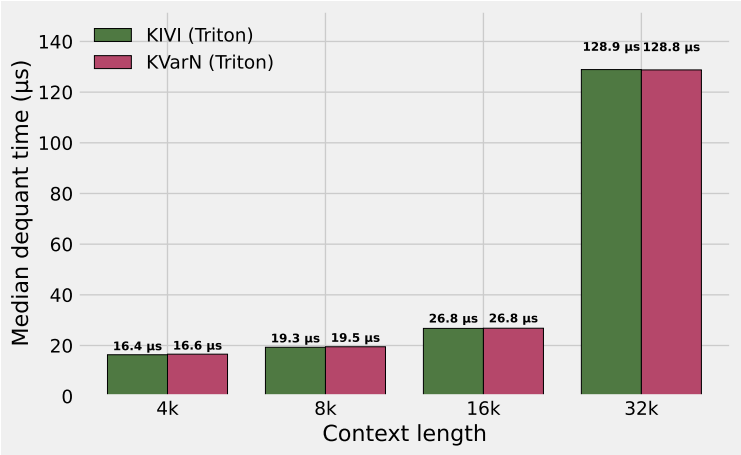


Figure 16: Median dequantization time per call for KIVI (single scale) vs. KVarN (dual scale, s_2 fused into the kernel) across context lengths. Lower is better.

Table 7: End-to-end decode throughput relative to the 16-bit baseline. Higher is better. Measurements use 256 distinct concurrent prompts and a 12K-token context window. GLM-4.5-Air-FP8 is a 106B-parameter MoE model evaluated with tensor parallelism over two GPUs. The KVarN 3-bit configuration uses K-4-bit and V-2-bit quantization. Notably, 2-bit KVarN has higher quality than 3-bit TurboQuant on all evaluations.

Method	**Bits**	**Model**		
		Qwen3-4B	**Qwen3-14B**	**GLM-4.5-Air-FP8**
Baseline	16	1.00×	1.00×	1.00×
TurboQuant[1]	3	0.70×	0.75×	0.59×
	4	0.79×	0.85×	0.69×
KVarN	2	**1.10**×	**1.20**×	**1.28**×
	3	1.03×	1.08×	1.18×
	4	0.96×	0.96×	1.03×

[1] Results obtained using the vLLM implementation.

We evaluate Qwen3-4B, Qwen3-14B, and GLM-4.5-Air-FP8. The latter is a 106B-parameter mixture-of-experts model evaluated with tensor parallelism across two GPUs. The reported bit widths refer to KV-cache precision: the 3-bit KVarN configuration uses 4-bit keys and 2-bit values.

At 3-bit precision, KVarN achieves throughput improvements of $1.47\times$, $1.44\times$, and $2.00\times$ over 3-bit TurboQuant on Qwen3-4B, Qwen3-14B, and GLM-4.5-Air-FP8, respectively. The 2-bit KVarN configuration achieves the highest throughput on all three models, exceeding the 16-bit KV-cache baseline by 10%, 20%, and 28%, respectively. At 4 bits, KVarN remains faster than TurboQuant on all three models, but exceeds the 16-bit baseline only on GLM-4.5-Air-FP8.

Q Computational Cost of Experiments

The dominant share of compute cost is due to the experiments on MATH500, AIME24, HumanEval and IFEval. The total cost to reproduce these are circa 50 GPU days on a GPU with 500 TFLOP at fp16 and 1.8 TB/s memory bandwidth.