STEPQUANT: WHEN AND WHERE ERRORS MATTER IN DELTA-RULE RECURRENT STATE QUANTIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Linear attention replaces growing KV caches with fixed-size recurrent states, yet these persistent states can become a substantial memory bottleneck under concurrent serving. Directly quantizing recurrent states to low precision often leads to severe accuracy degradation, as quantization errors propagate through successive state updates. We discover that the impact of these errors depends on two complementary dimensions: *temporally*, errors in long-lived memory can persist across many decoding steps; *spatially*, errors in different key rows affect model outputs differently, while state magnitudes vary substantially along both rows and columns. Motivated by these observations, we propose STEPQuant, a spatial-temporal post-training quantization framework for Delta-rule recurrent states. STEPQuant allocates precision according to error magnitude and memory lifetime, and jointly fits key-row and value-column scales based on state distributions and key-row impact on output error. Experiments on Qwen3.8-27B and Kimi-Linear-48B-A3B-Instruct across both long- and short-generation benchmarks show that STEPQuant closely matches FP32-state accuracy under a nominal 6-bit budget and outperforms uniform INT8 in its 4-bit configuration. Integrated into SGLang with optimized GPU kernels, 6-bit STEPQuant achieves over 5$\times$ recurrent-state compression and reduces total serving memory by up to 68.7%.

1 INTRODUCTION

Unlike conventional softmax attention, which maintains a KV cache that grows with sequence length, linear attention summarizes past tokens into a fixed-size recurrent state (Yang et al., 2023). Recent hybrid models, including Qwen3.8-27B (Qwen Team, 2026a) and Kimi-Linear-48B-A3B-Instruct (Team et al., 2025), combine gated Delta-rule recurrent memory (Yang et al., 2025) with standard attention to balance efficiency and performance. Despite its fixed size across context length, the recurrent state introduces a different memory bottleneck during serving. Each concurrent request requires a separate persistent state, while serving systems may reserve additional state slots for caching and scheduling. As a result, the memory footprint of the state pool *grows with concurrency even for short contexts*. In official SGLang deployment, the FP32 state pool of Qwen exceeds the memory footprint of its BF16 weights at 70 supported concurrent requests, as shown in Fig. 1(a). This growing memory overhead motivates efficient compression of recurrent states.

However, directly applying uniform quantization to recurrent states severely degrades model performance. Each Delta update operates on an approximate state and produces a newly quantized one, allowing quantization errors to propagate through subsequent decoding steps. Fig. 1(c) shows that mean accuracy drops sharply at 4 and 6 bits on both models, and a clear gap remains even at 8 bits.

To understand this failure, we investigate state quantization error from two complementary dimensions: *when* and *where* the error accumulates. First, *temporally*, recurrent states are updated and requantized at every decoding step, causing quantization errors to accumulate over time. Each update carries forward existing error while introducing new quantization error, and stronger gate retention allows these errors to persist longer. Our preliminary experiments discover that longer-lived state units tend to exhibit larger accumulated errors (Fig. 1(b)). Motivated by this observation, we introduce Lifetime-aware Bit Allocation, a mixed-precision quantization method that jointly considers the quantization error of each unit at different bit widths and its persistence over time. Under a fixed memory budget, it allocates higher precision to units with larger and longer-lived errors.

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.

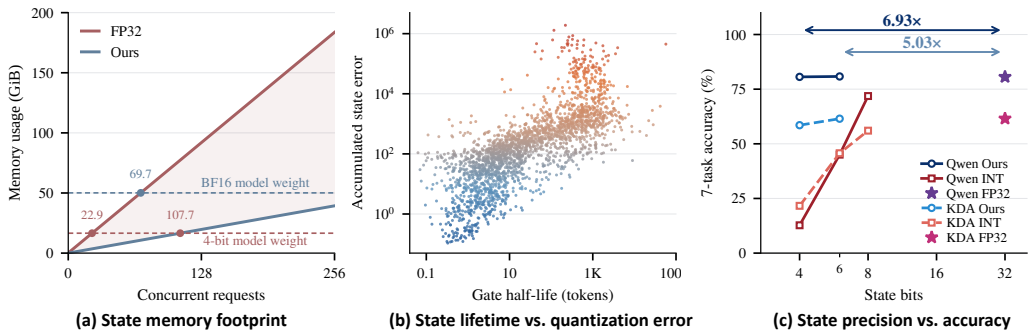

Figure 1: **Why recurrent state needs structured compression.** (a) Recurrent state memory grows with concurrent requests and can exceed model weight memory. STEPQuant substantially reduces memory cost at 6bit. (b) Heads with longer gate half-lives tend to accumulate larger state errors under uniform INT6 quantization across 2304 Qwen heads. (c) Mean accuracy across seven tasks for Qwen (solid) and KDA (dashed). STEPQuant achieves $6.93\times$ and $5.03\times$ compression of Qwen recurrent states at 4 and 6 bits, respectively, with negligible degradation in mean accuracy.

Second, *spatially*, the effect of quantization error also depends on where it occurs in the recurrent state. The spatial structure matters in two ways: errors of similar magnitude in different key rows can have different impacts on the model output, while state magnitudes vary substantially across both rows and columns (Fig. 2). Scaling along a single axis cannot accommodate both patterns as the state evolves during decoding. We therefore propose Key-Row-Aware Dual-axis Fitting to reduce spatial quantization error. Specifically, we assign separate scales to key rows and value columns. The row scales account for both the current magnitude of each row and its measured impact on output error, while the column scales are fitted with greater weight on rows where errors matter more.

Together, these temporal and spatial designs form **S**patial-**TE**m**P**oral **Quant**ization (STEPQuant), which compresses recurrent states throughout decoding. We evaluate STEPQuant on two strong hybrid models, Qwen3.8-27B and Kimi-Linear-48B-A3B-Instruct, across seven long-generation and six short-generation tasks. Under a nominal 6-bit budget, STEPQuant closely matches FP32-state accuracy on both models. Even with a 4-bit budget, STEPQuant outperforms uniform INT8 on both models, nearly matching FP32-state performance on Qwen while maintaining a modest gap on Kimi. With optimized kernels integrated into SGLang, 6-bit STEPQuant compresses recurrent-state memory by $5.03\times$ and $5.08\times$, reducing total memory consumption, including model weights, by 68.7% and 53.7% on Qwen and Kimi, respectively. Our contributions are summarized as follows:

- **Error analysis.** We analyze how quantization errors propagate through gated Delta-rule updates temporally and spatially, showing that both state lifetime and error location affect their impact.
- **Quantization method.** We propose STEPQuant, which combines lifetime-aware bit allocation with key-row-aware dual-axis fitting. Under a nominal 6-bit budget, STEPQuant closely matches FP32-state accuracy, while its 4-bit configuration outperforms uniform INT8 on both models.
- **Serving implementation.** We integrate STEPQuant into SGLang with optimized GPU kernels, achieving over $5\times$ recurrent-state compression and up to $2.91\times$ faster state updates.

2 Preliminaries

2.1 Gated Delta-Rule Linear Attention

Unlike softmax attention, which maintains a KV cache that grows with sequence length, recurrent linear attention summarizes past tokens in a fixed-size state matrix. For a single head in a given layer, we omit the layer and head indices and denote the state after token t by $S_t \in \mathbb{R}^{d_k \times d_v}$, with query $q_t \in \mathbb{R}^{d_k}$, key $k_t \in \mathbb{R}^{d_k}$, and value $v_t \in \mathbb{R}^{d_v}$. Gated DeltaNet (GDN) (Yang et al., 2025) and Kimi Delta Attention (KDA) (Team et al., 2025) update this state through a gated Delta rule:

$$S_t = D_t S_{t-1} + \beta_t k_t \big(v_t^\top - k_t^\top D_t S_{t-1}\big) = (I - \beta_t k_t k_t^\top) D_t S_{t-1} + \beta_t k_t v_t^\top, \qquad y_t = S_t^\top q_t, \quad (1)$$

where D_t controls memory retention and $\beta_t \in [0, 1]$ controls the write strength. The update first applies the retention gate D_t to the previous state, yielding the retained state $D_t S_{t-1}$. The current

key k_t retrieves $k_t^\top D_t S_{t-1}$ from the retained state. The residual $v_t^\top - k_t^\top D_t S_{t-1}$ is used to update the state along the key direction k_t. The head output (readout) is then computed as $y_t = S_t^\top q_t$.

The two architectures differ in their retention gate D_t. GDN uses a scalar gate per head, $D_t = \alpha_t I$, while KDA uses channel-wise gates $D_t = \mathrm{diag}(d_{t,1}, \cdots, d_{t,d_k})$. Both updates can be written as

$$S_t = A_t S_{t-1} + B_t, \qquad \text{where} \qquad A_t = (I - \beta_t k_t k_t^\top) D_t, \qquad B_t = \beta_t k_t v_t^\top. \tag{2}$$

Here, $A_t \in \mathbb{R}^{d_k \times d_k}$ transports and selectively modifies the previous memory, while $B_t \in \mathbb{R}^{d_k \times d_v}$ introduces the new value. The state contains $d_k \cdot d_v$ elements per head, independent of sequence length, but must persist across decoding steps for each active request.

2.2 Recurrent-State Quantization

Quantization maps floating-point values to a finite set of discrete levels, reducing storage requirements. In symmetric uniform quantization, each value x is encoded as a b-bit integer with a positive scale s shared within a quantization group. The reconstructed value is

$$\mathcal{Q}_{b,s}(x) = s \cdot \mathrm{clip}\left(\mathrm{round}\left(\frac{x}{s}\right), -q_b, q_b\right), \qquad q_b = 2^{b-1} - 1, \tag{3}$$

We focus on quantizing persistent recurrent states and also evaluate the combination with weight quantization. Let S_t denote the full-precision reference state obtained by recursively applying Eq. (2) without state quantization. At each decoding step, the model updates the reconstructed state in floating point and computes the output. The updated state is then quantized for storage:

$$X_t = A_t \hat{S}_{t-1} + B_t, \qquad \hat{y}_t = X_t^\top q_t, \qquad \hat{S}_t = \mathcal{Q}_t(X_t), \tag{4}$$

where $\mathcal{Q}_t$ quantizes X_t and returns its dequantized approximation $\hat{S}_t$.

3 Temporal Dimension: Lifetime-aware Bit Allocation

3.1 Lifetime-Dependent Error Accumulation

From Eq. 4, quantization error is recursively fed back through the state update. The following proposition characterizes how this error propagates across decoding steps. Proof is in Appendix A.

Proposition 1 (Conditional error propagation)**.** *For identical inputs and gates, let $E_t = \widehat{S}_t - S_t$ denote the accumulated error and $\varepsilon_t = \mathcal{Q}_t(X_t) - X_t$ the quantization error added at step t. Then*

$$E_t = (I - \beta_t k_t k_t^\top) D_t E_{t-1} + \varepsilon_t = A_t E_{t-1} + \varepsilon_t, \qquad \widehat{y}_t - y_t = E_{t-1}^\top A_t^\top q_t. \tag{5}$$

If $\|k_t\|_2 \le 1$, $0 \le \beta_t \le 1$, and $0 \preceq D_t \preceq I$, then $\|A_t\|_2 \le \|D_t\|_2 \le 1$.

According to Proposition 1, previously accumulated error is propagated through A_t, while quantization at step t introduces a new error. The transition A_t reduces existing error through two mechanisms. The retention gate D_t first attenuates the error carried over from the previous step. The Delta update then further reduces its component along the current key k_t through the factor $(I - \beta_t k_t k_t^\top)$, while leaving components orthogonal to k_t unchanged. We verify this effect experimentally in Appendix B.1. Thus, errors in directions rarely aligned with subsequent keys depend mainly on gate decay. When retention is close to one, these errors can persist for many decoding steps. Together, **these results identify memory lifetime as a key predictor of recurrent-state quantization risk.**

This analysis indicates that longer-lived memory should exhibit larger accumulated quantization error. To examine this assumption, we evaluate all recurrent heads of Qwen3.8-27B under uniform INT6 state quantization on C4. Fig. 1(b) shows that heads with longer gate half-lives exhibit larger accumulated state error, with Spearman's $\rho_S \approx 0.80$ (see Appendix B.5 for more details). This indicates that **recurrent heads with longer lives tend to result in larger quantization error.**

3.2 Lifetime-aware Bit Allocation

Mixed-precision quantization is widely used in LLMs to preserve accuracy by assigning more bits to sensitive components (Dettmers et al., 2022; 2023). For recurrent states, bit allocation should additionally account for memory lifetime, as quantization errors can persist across subsequent updates.

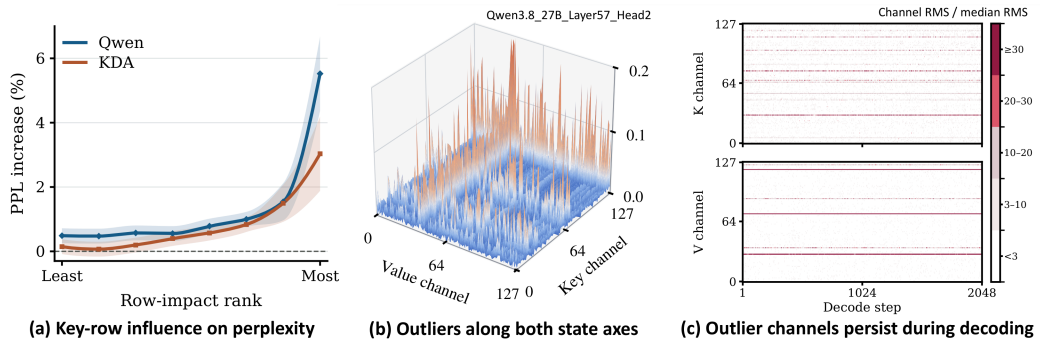

Figure 2: **Spatial structure of recurrent states.** (a) Key rows are ranked by readout impact and divided into eight groups. Quantizing one group at a time to INT4 generally causes larger perplexity increases for higher-impact groups. (b) A representative Qwen state exhibits obvious outliers in both key rows and value columns. (c) Channel RMS relative to the median across 2048 decoding steps. The outliers remain prominent throughout decoding.

Consequently, we propose Lifetime-aware Bit Allocation, which assigns precision under a fixed bit budget based on the persistence of quantization errors.

Lifetime-aware Bit Allocation operates on recurrent-state units: each unit u is an entire head in Qwen and a key row in KDA. We estimate $d_u(b)$, the reconstruction distortion of unit u quantized to b bits, and its mean log retention ℓ_u where the expectation is taken over calibration tokens. Here, $r_{t,u}$ is the retention gate for unit u: $r_{t,u} = \alpha_t$ for GDN and $r_{t,u} = d_{t,i}$ for key row i in KDA. A larger ℓ_u indicates stronger retention and a longer memory lifetime. Considering gate retention alone, the error retention factor after j updates is approximated by

$$\prod_{s=1}^{j} r_{t+s,u} \approx \exp(j\ell_u). \tag{6}$$

The squared error therefore decays by a factor of approximately $\exp(2j\ell_u)$, giving the lifetime weight over H steps:

$$L_u = \sum_{j=0}^{H-1} \exp(2j\ell_u). \tag{7}$$

Let n_u denote the number of state elements in unit u. Given an average bit budget $\bar{b}$, we select bit widths from the candidate set $\mathcal{B}_{\bar{b}}$ to minimize the lifetime-weighted reconstruction distortion:

$$\min_{b_u \in \mathcal{B}_{\bar{b}}} \sum_u L_u d_u(b_u) \quad \text{s.t.} \quad \sum_u n_u b_u \le \bar{b} \sum_u n_u. \tag{8}$$

This objective accounts for both *the magnitude of quantization error* and *how long it persists through subsequent updates*. Nevertheless, a small number of units remain difficult to quantize even with mixed-precision allocation. We therefore retain the highest-risk units in FP16 as sparse pivots.

In summary, Lifetime-aware Bit Allocation assigns higher precision to units with larger quantization errors and longer memory lifetimes. The pivot selection and bit allocation are determined during offline calibration and fixed across requests. Further details are provided in Appendix C.2 and D.

4 Spatial Dimension: Key-Row-Aware Dual-Axis Fitting

4.1 Key-row Impact on Readout Error

The temporal analysis above characterizes how quantization errors propagate across recurrent updates. We now examine their spatial distribution within the state matrix: errors of equal magnitude in different key rows can affect the readout differently. From Eq. (5), the readout error is

$$\Delta y_t = \hat{y}_t - y_t = E_{t-1}^\top A_t^\top q_t = \sum_i (A_t^\top q_t)_i E_{t-1,i,:}^\top, \tag{9}$$

We define $g_t = A_t^\top q_t \in \mathbb{R}^{d_k}$, where $g_{t,i}$ weights the contribution of the error in key row i, $E_{t-1,i,:}$, to the current readout error. For errors of equal norm confined to individual key rows, a larger $g_{t,i}^2$ implies a larger squared readout error. To account for variation across decoding steps, we define the row-impact score as

$$\omega_i = \mathbb{E}_{\text{cal}}[g_{t,i}^2]. \tag{10}$$

To validate this row-impact score, we rank key rows within each head by ω_i and divide them into eight groups. We quantize one group at a time to INT4 while keeping the remaining rows in full precision, and measure the resulting increase in perplexity. As shown in Fig. 2(a), groups with larger ω_i generally produce greater PPL degradation in both models. This observation suggests **using the row-impact score to guide quantization scale selection**, as described in Sec. 4.3.

4.2 Two-Axis State Geometry

Beyond differences in how key-row errors affect the readout, recurrent states exhibit another spatial property: large-magnitude outliers occur along both key rows and value columns. Unlike conventional LLM quantization settings that often target outliers along a single dominant axis (e.g., channels or tokens) (Xiao et al., 2023; Shao et al., 2024), we discover that the recurrent state matrix exhibits large-magnitude structures along both key rows and value columns, as shown in Fig. 2(b). We further examine how these magnitude patterns evolve throughout decoding. Fig. 2(c) shows that pronounced magnitude differences persist along both axes, with a small subset of rows and columns consistently exhibiting substantially larger magnitudes. Quantitative results show that the maximum-to-median RMS contrasts along the key-row and value-column axes are $10.3\times$ and $19.4\times$, respectively. Both exceed $3\times$ in 98.6% of sampled states. Together, these observations motivate **dual-axis scaling to accommodate the magnitude distributions of recurrent states**.

4.3 Key-Row-Aware Dual-axis Fitting

Sec. 4.1 and 4.2 identify two spatial properties of recurrent states: key rows differ in their impact on readout error, and state magnitudes vary substantially along both axes. To account for both, we propose Key-Row-Aware Dual-Axis Fitting, which combines calibrated row-impact scores with separate row and column scales. Specifically, we represent the updated state $X = X_t$ as:

$$\hat{X}_{ij} = r_i c_j z_{ij}, \tag{11}$$

where $r_i > 0$ and $c_j > 0$ are the scale factors for key row i and value column j, respectively, and z_{ij} is the low-bit integer. We incorporate the key-row impact factor into r_i, and the two factors allow the quantization independently to vary along the two state axes

Row factors. The row factor r_i should account for both (i) the current magnitude of a row and (ii) the impact of key row on readout error (Sec. 4.1). We first estimate the magnitude of key row i as

$$m_i = \frac{1}{d_v} \sum_j |X_{ij}|. \tag{12}$$

Rows with larger m_i require a wider quantization range, so the row factor should increase with m_i. In addition, key rows with larger impact scores ω_i are more vulnerable to quantization. Inspired by fractional-power smoothing in prior quantization methods (Xiao et al., 2023), we consider both effects using square-root scaling and a dimension-normalized w_i[1]:

$$r_i = m_i^{1/2} w_i^{-1/2}. \tag{13}$$

Thus, larger-magnitude rows receive a wider range, while higher-impact rows receive finer quantization resolution.

[1]We temper the dynamic range and normalize within each head as $w_i = \omega_i^{\gamma/2} / \mathrm{GM}_k(\omega_k^{\gamma/2})$, $\gamma = 0.25$, so that the geometric mean of w_i is one while preserving relative row importance, following (Box & Cox, 1964).

Column factors. Given the row factors $\{r_i\}$, we fit the column scales $\{c_j\}$ by minimizing the impact-weighted reconstruction error:

$$\min_{\{c_j>0\}} \sum_{i,j} w_i^2 \left(X_{ij} - r_i c_j z_{ij}\right)^2 . \tag{14}$$

This objective captures the remaining variation along value columns, while assigning larger penalties to reconstruction errors on high-impact key rows.

Quantization. Given the fitted scales, we quantize each scaled entry $X_{ij}/(r_i c_j)$ to the nearest representable level at its assigned precision b_i. The reconstructed state is then $\widehat{X}_{ij} = r_i c_j z_{ij}$. Implementation details of scale fitting and quantized-state storage are provided in Appendix C.3.

4.4 Kernel Implementation in SGLang

We implement STEPQuant as packed-state kernels integrated with SGLang's recurrent-state pool. Lifetime-Aware Bit Allocation and FP16 pivot selection are performed offline, adding no per-token allocation overhead and fixing the packed layout across requests. During decoding, we implement a kernel to fuse tilewise state reconstruction, the Delta update, and the current readout. This avoids a separate pass over the full state for reconstruction and reduces memory traffic. Once the head output is available, later layers continue processing the token while Key-Row-Aware Dual-Axis Fitting and packed writeback run on a separate CUDA stream. This overlaps scale updates with model computation and keeps the state compressed between tokens (see more details in Appendix F.1).

5 Experiments

5.1 Setup

Models and hardware. We evaluate Qwen3.8-27B (Qwen Team, 2026a) and Kimi-Linear-48B-A3B-Instruct (Team et al., 2025) with BF16 and 4-bit AWQ (Lin et al., 2024b) quantized weights on SGLang (Zheng et al., 2024), using four NVIDIA A800 GPUs. We use SGLang's default FP32 SSM-state precision as the full-precision baseline, alongside symmetric rowwise-absmax INT4/6/8 baselines. STEPQuant uses the fused state kernels described in Sec. 4.4.

Calibration. For all measured benchmarks, we use 32 WikiText-2 (Merity et al., 2016) training segments with 2048 tokens each to determine bit allocation, FP16 pivots, and row-impact scores.

Long-generation reasoning. We compare quantized models on seven reasoning benchmarks: LiveCodeBench v6 (Jain et al., 2025), EvalPlus (Liu et al., 2023), AIME 2026 (Dekoninck et al., 2026), MATH-500 (Lightman et al., 2024), HMMT February 2026 (Dekoninck et al., 2026), GPQA Diamond (Rein et al., 2023), and IFBench (Pyatkin et al., 2026). We generate 5, 5, 64, 4, 64, 8, and 4 samples per question, respectively, using temperature $T = 1.0$, top-$k = 20$, top-$p = 0.95$, and a maximum of 65536 generated tokens per sample.

Short generation. We also report results on six language understanding tasks: MMLU (Hendrycks et al., 2020), ARC-C (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), and LAMBADA (Paperno et al., 2016). We generate one answer per example with greedy decoding ($T = 0$) and score the generated answers rather than candidate likelihoods. This aligns the autoregressive decoding setting targeted by STEPQuant.

5.2 Accuracy across long and short tasks

For long reasoning tasks, Table 1 shows that 6-bit STEPQuant closely matches the FP32-state baseline on both Qwen (GDN) and Kimi (KDA). STEPQuant achieves mean accuracies of 80.59% on Qwen and 61.47% on Kimi, whereas uniform INT6 reaches only 45.04% and 45.70%, respectively. At 4 bits, uniform INT4 suffers substantial performance degradation, while STEPQuant achieves competitive performance across the seven tasks. The advantage also holds on the short-generation benchmarks in Table 2. Here, 4-bit STEPQuant trails FP32 by only 0.15 points on Qwen and 0.25 points on Kimi, whereas uniform INT4 degrades sharply and can fail to produce valid answers even on short tasks. These results show that recurrent states can be quantized below 8 bits with limited accuracy loss, a regime that the concurrent work DAMP identifies as challenging (see Appendix G).

Table 1: **Long-generation reasoning accuracy with BF16 weights (%).**

State	LCB v6	EvalPlus	AIME 26	MATH-500	HMMT	GPQA-D	IFBench	Avg.
Qwen3.8-27B								
FP32	85.31	84.87	87.71	97.60	74.24	80.81	53.67	80.60
INT8	72.23	80.26	78.54	97.00	58.71	75.25	41.00	71.86
INT6	30.43	69.00	33.96	87.40	16.67	51.52	26.33	45.04
INT4	7.87	31.55	0.00	34.00	0.00	4.04	11.67	12.73
STEPQuant@6bit	**85.42**	**85.54**	**87.24**	**97.35**	**73.30**	**80.87**	**54.42**	**80.59**
STEPQuant@4bit	**86.35**	**84.87**	**86.25**	**97.00**	**73.86**	**81.57**	**53.67**	**80.51**
Kimi-Linear-48B-A3B-Instruct								
FP32	54.52	76.75	68.33	93.25	45.08	69.70	23.00	61.52
INT8	52.61	75.46	51.88	92.45	35.61	62.88	21.25	56.02
INT6	42.37	71.96	22.71	88.00	17.42	57.83	19.58	45.70
INT4	28.82	46.86	0.00	45.80	0.00	14.58	15.33	21.63
STEPQuant@6bit	**54.86**	**76.57**	**67.71**	**93.60**	**46.21**	**68.43**	**22.92**	**61.47**
STEPQuant@4bit	**53.29**	**74.54**	**59.17**	**92.60**	**43.56**	**64.14**	**22.33**	**58.52**

Table 2: **Short-generation accuracy with BF16 weights (%).**

State	MMLU	ARC-C	OpenBookQA	HellaSwag	WinoGrande	LAMBADA	Avg.
Qwen3.8-27B							
FP32	82.25	96.93	95.40	93.15	90.06	68.91	87.78
INT8	82.19	96.16	94.60	92.00	87.69	64.89	86.25
INT6	78.86	94.28	92.80	87.72	79.79	61.50	82.49
INT4	78.80	73.63	77.60	60.51	46.57	57.31	65.74
STEPQuant@6bit	**82.23**	**96.67**	**95.80**	**93.01**	**89.19**	**68.35**	**87.54**
STEPQuant@4bit	**82.07**	**96.84**	**95.80**	**93.08**	**89.50**	**68.50**	**87.63**
Kimi-Linear-48B-A3B-Instruct							
FP32	71.21	91.64	88.60	67.42	55.88	35.40	68.36
INT8	70.67	91.98	88.40	66.78	55.80	33.17	67.80
INT6	67.73	87.54	83.40	61.77	41.28	26.37	61.35
INT4	57.36	69.37	72.80	19.88	17.36	16.67	42.24
STEPQuant@6bit	**71.21**	**90.61**	**89.40**	**68.77**	**57.70**	**35.20**	**68.82**
STEPQuant@4bit	**71.08**	**90.96**	**88.80**	**67.95**	**54.93**	**34.93**	**68.11**

5.3 Compatibility with W4 weights

We further evaluate STEPQuant with 4-bit AWQ-quantized weights, a setting more representative of practical deployment. From Table 3, 6-bit STEPQuant achieves seven-task average accuracies of 79.27% on Qwen and 58.62% on Kimi, only 0.05 and 0.33 points below their corresponding FP32-state baselines, respectively. At the lower 4-bit state budget, STEPQuant also maintains competitive performance. These results show that STEPQuant remains effective when combined with weight quantization. As weight memory decreases, recurrent states account for a larger fraction of the serving memory, making state compression increasingly important for memory-efficient deployment.

5.4 Component ablation

We evaluate the components on AIME 2026, GPQA Diamond, and LiveCodeBench v6 with BF16 weights under nominal 4- and 6-bit budgets. We also adapt the dual-axis state quantization (DSQ) component of Q-Mamba (Tianqi et al., 2025), originally designed for Mamba models, as a baseline. Spatial only retains uniform precision but applies key-row-aware dual-axis fitting. Temporal only uses calibrated mixed-precision allocation and FP16 pivots without spatial fitting, while Temporal w/o pivots removes the pivots. STEPQuant combines the spatial and temporal components. All quantization methods are evaluated under the same nominal bit budget (4 or 6 bits). Table 4 presents the Qwen results, with the corresponding Kimi results provided in Appendix E.4.

Component analysis. (i) *Spatial fitting.* At 4 bits, our spatial fitting substantially outperforms DSQ on Qwen (73.95% vs. 7.64%), with consistent improvements at 6 bits. This highlights the benefit of incorporating key-row impact into dual-axis quantization. (ii) *Pivot protection.* Protecting only

Table 3: **Reasoning performance of STEPQuant with 4-bit AWQ-quantized weights.**

State	LCB v6	EvalPlus	AIME 26	MATH-500	HMMT	GPQA-D	IFBench	Avg.
Qwen3.8-27B								
FP32	85.23	85.06	82.92	97.20	71.78	81.06	52.00	79.32
STEPQuant@6bit	**85.33**	**85.42**	**82.76**	**97.25**	**71.02**	**80.93**	**52.17**	**79.27**
STEPQuant@4bit	**83.98**	**84.69**	**85.42**	**96.40**	**70.83**	**81.06**	**50.50**	**78.98**
Kimi-Linear-48B-A3B-Instruct								
FP32	52.80	74.72	59.79	93.80	41.86	66.92	22.75	58.95
STEPQuant@6bit	**52.74**	**74.94**	**59.38**	**93.75**	**41.10**	**66.16**	**22.25**	**58.62**
STEPQuant@4bit	**51.94**	**74.17**	**51.88**	**92.20**	**40.91**	**63.38**	**22.17**	**56.66**

Table 4: **Component ablation on three long-reasoning tasks using Qwen3.8-27 BF16 weights.**

Nominal 6-bit budget

Variant	AIME	GPQA	LCB	Avg.
FP32	87.71	80.81	85.31	84.61
INT6	33.96	51.52	30.43	38.63
Q-Mamba@6bit	73.13	76.77	77.97	75.95
Spatial only	79.79	80.68	80.09	80.19
Temporal w/o pivots	60.42	71.97	69.00	67.13
Temporal only	74.58	79.67	72.64	75.63
STEPQuant@6bit	**87.24**	**80.87**	**85.42**	**84.51**

Nominal 4-bit budget

Variant	AIME	GPQA	LCB	Avg.
FP32	87.71	80.81	85.31	84.61
INT4	0.00	4.04	7.87	3.97
Q-Mamba@4bit	0.00	6.06	16.87	7.64
Spatial only	75.21	70.71	75.92	73.95
Temporal w/o pivots	0.00	6.57	11.94	6.17
Temporal only	3.96	11.62	23.03	12.87
STEPQuant@4bit	**86.25**	**81.57**	**86.35**	**84.72**

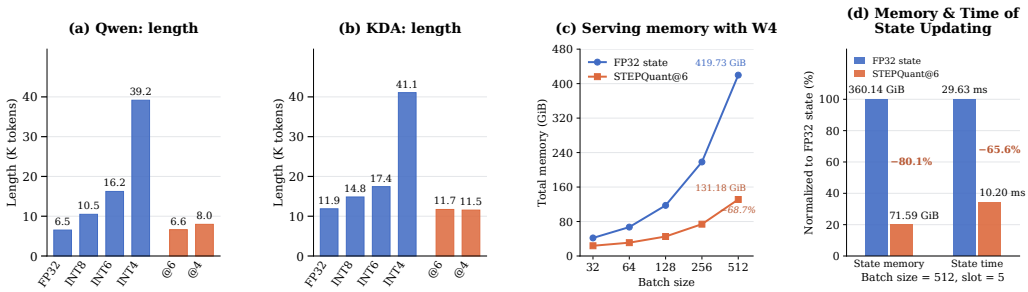

Figure 3: **Generation length and serving efficiency of STEPQuant.** (a–b) Mean generated tokens across seven tasks, weighting tasks equally. Blue bars show FP32 and uniform INT8/6/4. Orange bars show STEPQuant@6bit/@4bit (marked @6/@4). Lengths include thinking, incorrect answers, and capped outputs. (c) Total serving memory of Qwen with W4 weights across different batch sizes. (d) Normalized recurrent-state memory and updating time of Qwen at batch size 512.

1.39% of Qwen heads with FP16 pivots improves the 4-bit three-task average by 6.70 points and the 6-bit AIME accuracy by 14.16 points. This demonstrates that protecting a small fraction of high-risk units can substantially improve accuracy. (iii) *Combining both components.* At 4 bits, STEPQuant achieves 84.72% on Qwen, outperforming both Spatial only (73.95%) and Temporal only (12.87%). At 6 bits, STEPQuant achieves 84.51%, compared with 80.19% and 75.63% for the individual components, closely matching FP32 (84.61%). These results demonstrate the complementary benefits of our spatial and temporal components. Similar trends are observed on Kimi under both bit budgets.

5.5 Generation length on seven long-generation tasks

Prior works (Liu et al., 2025; Lotfi et al., 2026) have observed that quantized models may exhibit overthinking, producing excessively long outputs without improving reasoning accuracy. We therefore examine generation length to further understand the behavior of STEPQuant on reasoning tasks. As presented in Fig. 3(a)(b), uniform quantization substantially increases output length. For example, Kimi generates an average of 63.40K tokens on AIME and 64.22K on HMMT under 4-bit quantization, approaching the 65536-token limit while achieving near 0 accuracy on both tasks. This suggests that the increased generation length does not translate into effective reasoning under aggressive uniform quantization. In contrast, STEPQuant maintains output lengths close to those of the FP32-state model, consistent with its preserved accuracy on long reasoning tasks.

5.6 Efficiency Evaluation

We evaluate the serving efficiency of STEPQuant using SGLang on A800 GPUs. Fig. 3(c) reports the total serving memory of Qwen with W4 weights across different batch sizes. At a batch size of 512, STEPQuant@6bit reduces total memory from 419.73 to 131.18 GiB, a 68.7% reduction. Beyond the overall memory savings, we further examine recurrent-state memory and state-update time. Fig.3(d) shows that STEPQuant@6bit reduces recurrent-state memory by 80.1% (5.03× compression) and state-update time by 65.6% (2.91× faster). Additional results, including the evaluation protocol, state memory and update time on Kimi, and full-model decode throughput on both models, are provided in Appendix F.

6 Related Work

6.1 Linear Attention Transformers

Standard self-attention incurs quadratic computation in sequence length and maintains a growing KV cache during autoregressive decoding (Zhang et al., 2023). Linear attention can instead summarize past key-value interactions in a fixed-size recurrent state, enabling linear-time sequence processing and constant state storage with respect to sequence length. Subsequent works improve memory management and computational efficiency: RetNet (Sun et al., 2023) introduces multiscale retention with parallel and recurrent formulations, while Gated Linear Attention (Yang et al., 2023) uses data-dependent gates to control information retention. Related state-space models, including Mamba (Gu & Dao, 2024) and Mamba-2 (Dao & Gu, 2024), also combine recurrent inference with efficient training algorithms. Gated DeltaNet (GDN) (Yang et al., 2025) combines head-wise forgetting with delta-rule updates to selectively modify associations stored in its matrix-valued state. Kimi Delta Attention (KDA) (Team et al., 2025) extends this mechanism with channel-wise forgetting and serves as the main component of Kimi Linear and K3 (Team et al., 2026), which interleave KDA and softmax attention layers. Qwen families (Qwen Team, 2026b;c;d;a) also adopt the hybrid GDN architecture to deliver high-throughput inference with minimal latency overhead. These advances improve how recurrent states store and update information, but fixed state size does not eliminate storage costs: total state memory scales with batch size, layer count, and state dimensions. We study low-bit quantization of GDN recurrent states to reduce this inference memory footprint.

6.2 Post-training Quantization

Post-training quantization (PTQ) reduces model memory by representing weights in low-bit formats (Lin et al., 2024b; Frantar et al., 2022; Shao et al., 2024), while jointly quantizing weights and activations can accelerate inference through low-bit kernels (Ashkboos et al., 2024; Lin et al., 2024a). PTQ can also reduce the growing KV cache in long-context inference (Hooper et al., 2024; Zandieh et al., 2026). For example, KIVI(Liu et al., 2024) quantizes keys and values along different axes. For state-space models, Quamba (Chiang et al., 2025b) and MambaQuant (Xu et al., 2025) target weight and activation quantization, while Quamba2 (Chiang et al., 2025a) additionally quantizes cached recurrent states to 8 bits. Q-Mamba (Tianqi et al., 2025) addresses state-cache quantization through dual-axis scaling and selectivity reconstruction. Quantization of GDN recurrent states remains less explored. Concurrent work, DAMP (Zhang et al., 2026), uses decay-based persistence to select key channels for FP16 protection while quantizing the remaining channels to INT8, reporting preserved accuracy at 9.9 bits per state value in its evaluated settings. Our work demonstrates that lower precision is feasible: STEPQuant quantizes recurrent states in KDA and Qwen3.8 models to 6 bits with negligible accuracy degradation on the evaluated benchmarks.

7 Conclusion

We study low-bit quantization of recurrent states in Delta-rule models and show that quantization errors are shaped by both temporal persistence and spatial structure. Based on these observations, we propose STEPQuant, combining lifetime-aware bit allocation with Key-Row-Aware Dual-axis Fitting. Experiments across long- and short-generation tasks show that STEPQuant enables accurate low-bit state quantization and substantially reduces recurrent-state memory for concurrent serving.

AI USE STATEMENT

AI assistants were used only for language polishing, LaTeX checking, and debugging assistance. All scientific ideas, methodological designs, experiments, analyses, and conclusions were developed, conducted, and verified by the authors.

REPRODUCIBILITY STATEMENT

We describe every component needed to reproduce our experiments. We provide the recurrent-state update and quantization definitions in Section 2, the precision-allocation objective in Section 3.2, and the spatial fitting procedure in Section 4.3. Appendix A gives the proof of conditional error propagation and defines the calibration statistics. Appendices C.2 and C.3 describe the calibration data, candidate precisions, FP16 pivots, allocation procedures, and quantized decode workflow. Section 5.1 specifies the models, benchmarks, and generation settings, while Appendix F documents the SGLang integration, hardware configurations, decode timing protocol, and memory accounting. Additional per-task generation lengths and component ablations are provided in Appendix E.

REFERENCES

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.

George EP Box and David R Cox. An analysis of transformations. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 26(2):211–243, 1964.

Hung-Yueh Chiang, Chi-Chih Chang, Natalia Frumkin, Kai-Chiang Wu, Mohamed S Abdelfattah, and Diana Marculescu. Quamba2: A robust and scalable post-training quantization framework for selective state space models. *arXiv preprint arXiv:2503.22879*, 2025a.

Hung-Yueh Chiang, Chi-Chih Chang, Natalia Frumkin, Kai-Chiang Wu, and Diana Marculescu. Quamba: A post-training quantization recipe for selective state space models. In *International Conference on Learning Representations*, volume 2025, pp. 101328–101354, 2025b.

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

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. *arXiv preprint arXiv:2605.00674*, 2026.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35: 30318–30332, 2022.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*, 2023.

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

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

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

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.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

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 *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Haokun Lin, Haobo Xu, Yichen Wu, Jingzhi Cui, Yingtao Zhang, Linzhan Mou, Linqi Song, Zhenan Sun, and Ying Wei. Duquant: Distributing outliers via dual transformation makes stronger quantized llms. *Advances in Neural Information Processing Systems*, 37:87766–87800, 2024a.

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

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

Ruikang Liu, Yuxuan Sun, Manyi Zhang, Haoli Bai, Xianzhi Yu, Tiezheng Yu, Chun Yuan, and Lu Hou. Quantization hurts reasoning? an empirical study on quantized reasoning models. *arXiv preprint arXiv:2504.04823*, 2025.

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

Sanae Lotfi, Polina Kirichenko, Steven Li, and Zechun Liu. Quantized reasoning models think they need to think longer, but they do not. *arXiv preprint arXiv:2606.00206*, 2026.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2381–2391, 2018.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1525–1534, 2016.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026. URL https://openreview.net/forum?id=yfYgwjj5F8.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026a. URL https://qwen.ai/blog?id=qwen3.8.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026b. URL https://qwen.ai/blog?id=qwen3.5.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model, April 2026c. URL https://qwen.ai/blog?id=qwen3.6-27b.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026d. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

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

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

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

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, et al. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Chen Tianqi, Yuanteng Chen, Peisong Wang, Weixiang Xu, Zeyu Zhu, and Jian Cheng. Q-mamba: Towards more efficient mamba models via post-training quantization. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10594–10610, 2025.

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

Zukang Xu, Yuxuan Yue, Xing Hu, Zhihang Yuan, Zixu Jiang, Zhixuan Chen, Jiangyong Yu, Chen Xu, Sifan Zhou, and Dawei Yang. Mambaquant: Quantizing the mamba family with variance aligned rotation methods. In *International Conference on Learning Representations*, volume 2025, pp. 33231–33250, 2025.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025.

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

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

Tao Zhang, Jianchao Tan, Pingwei Sun, Yanqi Yu, Zixu Jiang, Yuchen Xie, Xunliang Cai, and Ziqian Zeng. Damp: Decay-aware mixed-precision recurrent-state quantization. *arXiv preprint arXiv:2608.27513*, 2026.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in neural information processing systems*, 36:34661–34710, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

A Recurrent-State Error Propagation and Calibration Statistics

This appendix proves the conditional error propagation in Section 3 and relates cumulative squared output error to the row-impact scores used by Key-Row-Aware Dual-axis Fitting in Section 4.3. It also defines the calibration statistics used by STEPQuant's temporal and spatial components.

A.1 Proof of Proposition 1

Subtracting the reference recurrence from the quantized update gives

$$\begin{aligned} E_t &= \mathcal{Q}_t(A_t\widehat{S}_{t-1} + B_t) - (A_t S_{t-1} + B_t) \\ &= A_t(\widehat{S}_{t-1} - S_{t-1}) + \varepsilon_t. \end{aligned}$$

At step t, the output is read from X_t before $\mathcal{Q}_t(X_t)$ is stored, so $\widehat{y}_t - y_t = (A_t E_{t-1})^\top q_t$. Let $P_t = I - \beta_t k_t k_t^\top$. Under the proposition's conditions, $0 \preceq P_t \preceq I$, so $\|P_t\|_2 \le 1$. Therefore

$$\|A_t\|_2 = \|P_t D_t\|_2 \le \|P_t\|_2 \, \|D_t\|_2 \le \|D_t\|_2 \le 1.$$

A.2 Cumulative squared output error

Condition on the same keys, values, queries, and gates in both paths, as in Proposition 1. Let $\Phi_{s,t} = A_s \cdots A_{t+1}$ transport a state perturbation from step t to s. Over a horizon of H future reads, an isolated quantization error Z produces cumulative squared output error

$$\sum_{s=t+1}^{t+H} \|Z^\top \Phi_{s,t}^\top q_s\|_2^2 = \mathrm{tr}(Z^\top W_{t,H} Z), \quad W_{t,H} = \sum_{s=t+1}^{t+H} \Phi_{s,t}^\top q_s q_s^\top \Phi_{s,t}. \tag{15}$$

For a single perturbation Z at time t and no subsequent quantization errors, the state perturbation after transition s is $\Phi_{s,t} Z$. The output perturbation is $Z^\top \Phi_{s,t}^\top q_s$. Using $\left\|Z^\top u\right\|_2^2 = \mathrm{tr}(Z^\top u u^\top Z)$ and summing proves Equation 15. Here $W_{t,H}$ is the readout-error Gram matrix and tr denotes the matrix trace. This matrix is positive semidefinite.

For a single future read, $H = 1$, this matrix reduces to

$$W_{t,1} = g_{t+1} g_{t+1}^\top, \qquad g_{t+1} = A_{t+1}^\top q_{t+1}.$$

A unit-Frobenius-norm perturbation confined to key row i therefore produces squared output error at the next step $g_{t+1,i}^2$. Both Qwen and KDA calibrate $\omega_i = \mathbb{E}_{\mathrm{cal}}[g_{t,i}^2]$, the mean diagonal of this one-step readout-error matrix. STEPQuant's spatial component normalizes ω_i into the row-impact factors w_i. These determine the row scales $r_i = m_i^{1/2} w_i^{-1/2}$ and the squared weights w_i^2 in the column-fitting objective (Section 4.3 and Appendix C).

With quantization errors ε_r introduced at successive steps and $E_0 = 0$, the accumulated error is

$$E_t = \sum_{r=1}^{t} \Phi_{t,r} \varepsilon_r, \qquad \Phi_{t,t} = I.$$

The squared norm includes cross terms between propagated quantization errors.

A.3 Calibration Statistics for Temporal and Spatial Components

STEPQuant's temporal component uses Lifetime-aware Bit Allocation and sparse FP16 pivots, while its spatial component uses Key-Row-Aware Dual-axis Fitting. The statistics below connect their calibration.

Mean log retention. For allocation unit u, define

$$\ell_u = \mathbb{E}_{\mathrm{cal}}[\log r_{t,u}], \qquad L_u = \sum_{j=0}^{H-1} \exp(2j\ell_u). \tag{16}$$

The expectation is taken over calibration tokens. For Qwen, u is a state head and $r_{t,u} = \alpha_{t,u}$; for KDA, u is a key row and $r_{t,u} = d_{t,u}$. The lifetime weight L_u is applied to the candidate-format distortion after that distortion is averaged over state samples.

Let $\mathcal{C}$ contain the sampled reference states, with each batch element counted as a separate state sample. Let $S_{s,u}$ be the reference state of unit u at sample s, and $\widehat{S}^{(b)}_{s,u}$ its candidate-format reconstruction at precision b.

Qwen reconstruction distortion. For Qwen, u is an entire head and $n_u = d_k d_v$. Its candidate-format calibration uses the spatial component's row-impact factors $w_{u,i}$:

$$d_u(b) = \frac{1}{|\mathcal{C}|\, n_u} \sum_{s\in\mathcal{C}} \sum_{i,j} w^2_{u,i} \left(\widehat{S}^{(b)}_{s,u,ij} - S_{s,u,ij}\right)^2. \tag{17}$$

The squared row-impact factors $w^2_{u,i}$ weight the reconstruction error. Weighted squared error is averaged over the elements within each head and then over state samples; the temporal lifetime weight L_u is applied afterward.

KDA reconstruction distortion. For a KDA key row, $n_u = d_v$ and the calibrated distortion is the unweighted per-element MSE:

$$d_u(b) = \frac{1}{|\mathcal{C}|\, n_u} \sum_{s\in\mathcal{C}} \left\| \widehat{S}^{(b)}_{s,u} - S_{s,u} \right\|_2^2. \tag{18}$$

KDA samples states every eight recurrent updates, averages squared error over batch elements and value coordinates, and then averages over the sampled update positions. Once FP16 pivots are fixed, their rows do not participate in fitting the shared column scales for integer rows.

Thus, Qwen's temporal allocation uses a distortion measure that already accounts for spatial row impact. In KDA, the allocation MSE is unweighted, but the candidate reconstruction is produced with the spatial codec and the shared column fit excludes FP16 pivot rows. The two components have distinct roles, while sharing the calibrated state representation. Within each model, all allocation units have the same n_u. Converting per-element MSE to total squared error therefore multiplies the allocation objective by a model-specific constant and does not change the selected precision assignment.

B Analysis of Temporal Error Accumulation and Spatial Row Impact

These controlled experiments support the temporal analysis in Section 3 and the spatial analysis in Sections 4.1 and 4.2. They use reference input streams to distinguish error persistence from differences in key-row impact and state geometry.

B.1 Effect of Delta Feedback on Quantization Error

For GDN, the directional contraction is

$$\|\alpha(I - \beta k k^\top)E\|_F^2 = \alpha^2[\|E\|_F^2 - \beta(2 - \beta\|k\|_2^2)\|k^\top E\|_2^2]. \tag{19}$$

Let $P = I - \beta k k^\top$. Expanding $P^\top P$ gives $I - \beta(2 - \beta\|k\|_2^2)kk^\top$, which proves Equation 19. For KDA, the corresponding subtraction is applied to $D_t E_{t-1}$, retaining the native order of decay and correction. The mechanism does not require the state transition to amplify perturbations: repeated injections can accumulate under a non-expansive transition.

The exact-read oracle replaces only the state inside the Delta residual:

$$\widetilde{S}^{\text{oracle}}_t = D_t \widehat{S}^{\text{oracle}}_{t-1} + \beta_t k_t (v_t^\top - k_t^\top D_t S_{t-1}).$$

Subtracting the reference update leaves $D_t E^{\text{oracle}}_{t-1}$. This intervention removes the correction of the path's own state error. With continuous quantization it also changes future rounding errors. Therefore we use a matched single-injection experiment to isolate feedback.

Each of four fixed C4 (Raffel et al., 2020) Qwen trajectories provides native keys, values, queries, and gates for 8,192 updates. We use the deterministic Cartesian subset of global model-layer indices $\{0, 8, 16, 24, 32, 40, 48, 62\}$ and global heads $\{0, 12, 24, 36\}$, giving 32 heads fixed before replay and independent of quantization outcomes. We inject one quantization error at step 256 and perform the remaining 7,936 updates without further quantization. Both paths start from the identical error. We sum squared output error over heads, trajectories, and subsequent steps. Relative to native Delta feedback, the exact-read oracle increases this error by $26.82\times$ for INT6 and $18.65\times$ for INT8; both ratios exceed one on each of the four trajectories.

B.2 Prefill and Decode Prediction Drift

An uninterrupted native prefill evolves an uncompressed state and packs it once at the boundary. Decode then reads and rewrites the compressed state after each token. This probe measures prediction drift caused by repeated compressed-state updates.

On four held-out C4 streams per model, each with a 2,048-token native prefill and 6,144 forced decode updates, prefill predictions match the FP32-state reference. Mean excess NLL (quantized minus reference) over the first and last 256 compressed-read predictions rises from 0.102 to 2.102 for Qwen INT6, and from 0.0228 to 0.1422 for KDA INT6. Across all 6,144 decode predictions, STEPQuant@6bit has mean excess NLL of 0.0011 on Qwen and -0.0120 on KDA.

B.3 Readout Error at Fixed Precision

For Qwen, the calibrated row-impact factor satisfies $w_i \propto \omega_i^{1/8}$, where $\omega_i = \mathbb{E}[(A_t^\top q_t)_i^2]$ includes the immediate Delta transition and readout (Section 4.3). Within a head, the normalization constant cancels in row-impact ratios. We recover a proportional row-impact score as w_i^8 from the frozen full-precision calibration record, retaining its numerical floor, and compute the interpolated P90/P10 ratio across 128 key channels. Across all 2,304 heads, the median ratio is 15.95. The 10th and 90th percentiles across heads are 3.74 and 79.84. All channels within each such head share exactly the same scalar gate. This difference is thus additional to gate lifetime and directly motivates the spatial component's within-head row-impact factor.

Figure B1d isolates row impact at fixed assigned precision. A group is one integer Qwen head, or the integer rows sharing both head and bit width in KDA, using frozen STEPQuant@6bit assignments. FP16 pivots are excluded. Requiring at least eight rows per group retains 2,272 Qwen groups (290,816 rows) and 1,567 KDA groups (80,333 rows). 1,075 KDA integer rows in smaller groups are excluded, without filtering on measured error. Qwen rows in a group share a scalar gate. KDA rows can have different keywise gates despite sharing their assigned bit width.

On reference-state traces, we inject a unit-Frobenius-norm perturbation E_i confined to key row i. With $A_t = (I - \beta_t k_t k_t^\top) D_t$ and $g_t = A_t^\top q_t$, its isolated next-read squared error is exactly $\|E_i^\top g_t\|_2^2 = g_{t,i}^2$. We average 32 probe positions following 128 replay updates over four Qwen and two KDA C4 streams. Direct propagation checks agree with this identity within 7.1×10^{-7} relative error.

Rows are ranked within each group using independent WikiText calibration: frozen w_i^8 for Qwen and one-step transported-query row-impact scores for KDA. Each row's held-out read error is divided by its own group's mean. We split each ranked group into eight near-equal bins and average bin means with equal group weight. Points are measured means at the averaged percentile centers. Curves use shape-preserving interpolation without extrapolation. The highest-to-lowest bin ratios are 117.4 for Qwen and 35.1 for KDA. These within-group contrasts motivate Key-Row-Aware Dual-axis Fitting even after lifetime-aware precision allocation.

B.4 Key-Row Quantization and Perplexity

The perplexity experiment in Figure 2(a) tests whether calibrated row impact predicts the effect of state quantization on model outputs (Section 4.1). Within each head, we rank key rows by the independently calibrated ω_i and divide them into eight near-equal groups. We quantize one ranked group at a time to INT4 across the model, keeping all other rows in full precision. Perplexity is measured on eight 1,024-token streams per model and compared with the FP32-state reference.

Markers show the measured bin results, and curves use shape-preserving interpolation. Higher-impact groups generally cause larger perplexity increases in both models. Unlike the equal-norm probe above, this experiment measures the effect of quantizing actual state values on the model's predictive distribution.

B.5 Gate Half-Life and Accumulated INT6 State Error

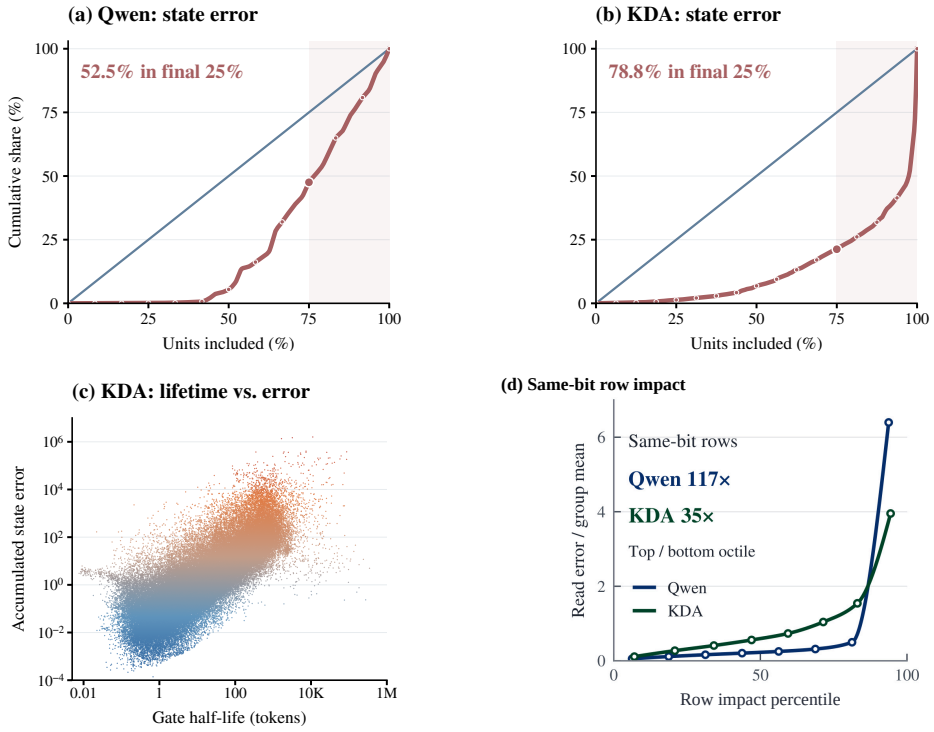

Figure B1: **Gate lifetime and key-row impact of recurrent state.** (a–b) Cumulative INT6 squared error (red) and unit count (blue), ordered by increasing gate half-life within each layer or head. The longest-lived quarter accounts for 52.5% of Qwen error and 78.8% of KDA error across all 2,304 heads and 81,920 channels. (c) KDA half-life versus accumulated INT6 error for 79,864 channels after excluding near-zero errors; color encodes error. (d) Held-out equal-norm readout-error contrast across row-impact octiles within fixed-precision groups, using independent WikiText ranking.

Figure 1b shows the Qwen heads, and Figure B1c shows the corresponding KDA key channels. Both scatters measure state error under uniform INT6. Full-population Spearman correlations are 0.8004 and 0.8017, respectively. For legibility, the KDA scatter omits 2,056 channels (2.51%) with squared error below 10^{-15}. The remaining 79,864 channels are shown on logarithmic axes. A Qwen head has 16,384 state entries and a KDA key row has 128, so raw error magnitudes across models are not directly comparable.

For Figure B1, frozen WikiText calibration estimates ℓ_u as defined in Appendix A.3, and gives the gate half-life $\tau_u = \log(2)/(-\ell_u)$. This gate-only statistic describes forgetting. The Delta correction remains part of the measured recurrence. The error experiment uses four frozen C4 sequences per model with their model-specific tokenizers. After a two-token FP32 boundary, a separate INT6 trajectory follows the reference keys, values, and gates for 2,046 decode updates. Every key row uses an absmax/31 scale stored in FP16, clamped to $[2^{-14}, 65504]$, and nearest-integer codes in $[-31, 31]$. Quantization is applied at the boundary and after every recurrent update. For unit u we record

$$D_u = \sum_{r=1}^{4} \sum_{t=1}^{2046} \left\| (\widehat{S}_{r,t} - S_{r,t})_u \right\|_F^2 .$$

A Qwen unit is a whole head matrix. A KDA unit is a key row containing 128 value components. These units partition each state, so their squared errors sum exactly. We sort by lifetime within each group, sum D_u across groups at each rank, and then accumulate from shortest to longest. The error curve is normalized by its model-wide total. No per-layer or per-head normalization precedes pooling. The longest quarter contains 12 heads per Qwen layer or 32 channels per KDA head. Shape-preserving cubic interpolation passes through every measured cumulative-rank point.

Qwen replays captured native FP32 inputs, checking reference readouts against the capture. KDA executes its native recurrent kernel on a separate shadow state. These controlled mechanism replays isolate persistent-state distortion along reference input streams.

B.6 Lifetime Rankings and Dual-Axis State Geometry

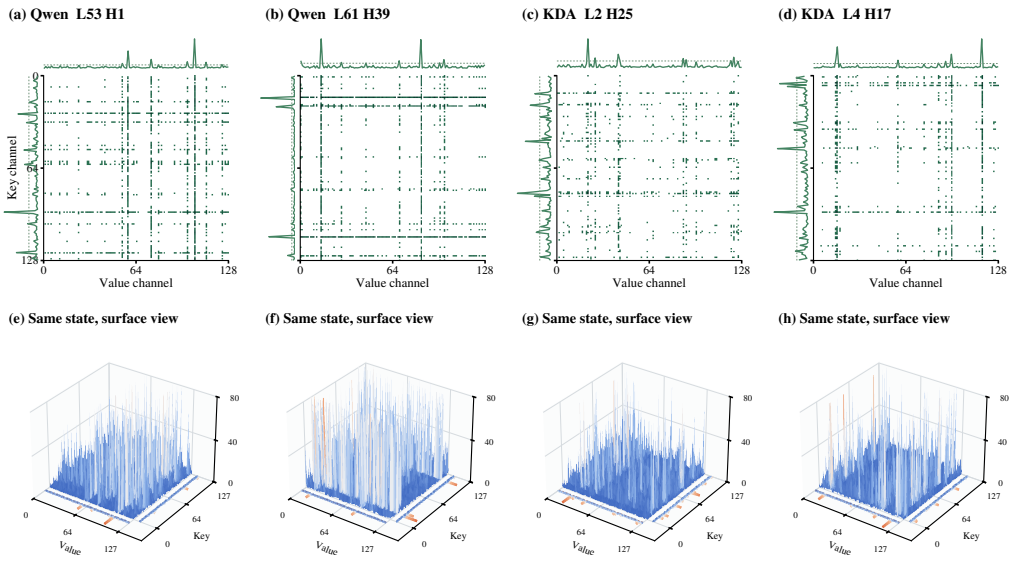

Figure B2: **Dual-axis state geometry across Qwen and KDA.** Four native-order 128×128 states (Qwen, Qwen, KDA, KDA). Top: entries below $10\times$ the matrix-median absolute magnitude are white; larger entries use graded greens. Marginal traces show row and column RMS relative to their axis medians. Bottom: matching surfaces use the same style as the Qwen surface in Figure 2(b). Floor traces show the same normalized RMS profiles, rescaled for display; heights are capped at $80\times$. Each state has seven or eight rows and columns above $5\times$ their axis-median RMS.

Figure B3 compares gate-lifetime ordering under WikiText-2 (Merity et al., 2016), C4, and LiveCodeBench (Jain et al., 2025) text. The frozen WikiText calibration is independent of the task streams. The BF16 Qwen gate observer covers all 2,304 recurrent heads using four archived 2,048-token C4 streams and the first four unique sample-0 LiveCodeBench trajectories in sorted archive order. The latter combine original prompts and FP32-reference generated token IDs, truncated at 2,048 tokens without padding, for 6,218 observed tokens per head. KDA covers all 81,920 key channels. Its C4 inputs are four 2,048-token evaluation streams. Its LiveCodeBench inputs follow the same selection rule, combining original prompts with archived KDA W4 FP32-state generated tokens; the BF16 KDA gate observer processes 8,192 C4 and 3,275 LiveCodeBench tokens per layer.

For each text source, we compute token-weighted mean log gate retention and rank the resulting half-lives from shortest to longest across all units of each model. Figure B3 plots these global ranks as percentiles. Full-population Spearman correlations with WikiText are 0.989 (C4) and 0.985 (LiveCodeBench) for Qwen, and 0.991 and 0.982 for KDA. For Qwen, the short/middle/long classes defined by within-layer ranks 1–12, 13–36, and 37–48 retain their WikiText labels for 93.6% of heads on C4 and 90.1% on LiveCodeBench.

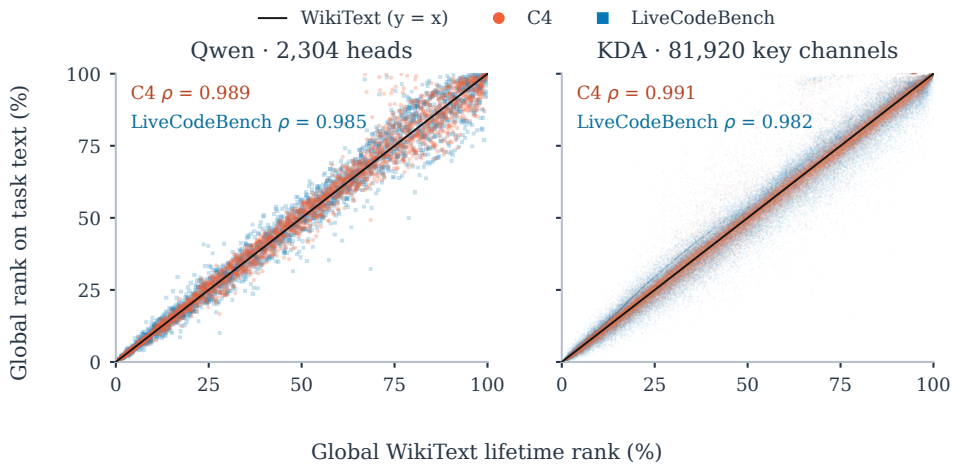

Figure B3: **Global lifetime rankings remain similar across text sources.** Qwen (2,304 heads) and KDA (81,920 key channels) are ranked separately, from shortest to longest gate half-life. Each point compares its global WikiText rank percentile with its C4 (orange-red) or LiveCodeBench (sky blue) percentile. The thin black diagonal is WikiText compared with itself ($y = x$). Spearman coefficients use the complete population in each panel.

Figure B2 uses two Qwen C4 snapshots at step 2,048 from distinct layers and heads. The two KDA snapshots are replayed from captured C4 keys, values, gates, and update coefficients at steps 128 or 256, also from distinct layers and heads. The native channel order and all matrix entries are retained. We chose the four examples by the $5\times$ RMS criterion in the caption. Across the full KDA capture of 3,840 snapshots at steps 128, 256, and 512, median maximum-to-median RMS contrasts are $12.7\times$ for key rows and $5.4\times$ for value columns; both contrasts exceed $3\times$ in 3,262 snapshots (84.9%).

For the Qwen state-geometry analysis in Section 4.2, FP32 state matrices are reconstructed from native recorded inputs at positions 128, 512, and 2,048. Define $r_i^{\mathrm{rms}} = \sqrt{\frac{1}{128}\sum_j S_{ij}^2}$ and $c_j^{\mathrm{rms}} = \sqrt{\frac{1}{128}\sum_i S_{ij}^2}$. The outlier contrasts are $\max_i r_i^{\mathrm{rms}} / \operatorname{median}_i r_i^{\mathrm{rms}}$ and the analogous column ratio. Medians use linear interpolation. Four C4 streams provide 27,648 matrices and two AIME streams provide 13,824. These repeated snapshots describe state geometry and are not independent model replicates. On C4, the median contrasts are $10.3\times$ for key rows and $19.4\times$ for value columns, and both exceed $3\times$ in 98.6% of snapshots, as reported in Section 4.2. The representative Qwen surface in Figure 2(b) uses global layer 57, head 2, C4 stream 2 at step 2,048. It was selected for clearly visible extended row and column ridges whose intersection is also an outlier. All 128×128 entries retain their native order. The linear height axis ends at 0.2, with larger values clipped for display.

The time-resolved row and column profiles in Figure 2(c) apply the same RMS normalization at each recorded decode update. They show how large-magnitude channels persist over the trajectory; the snapshot-based population statistics use all recorded matrices.

C Calibration, Precision Allocation, and Decode Procedure

This appendix complements STEPQuant's Lifetime-aware Bit Allocation, Key-Row-Aware Dual-axis Fitting, and execution steps in Sections 3.2 and 4.3 and Appendix C.3.

C.1 Row-Impact Calibration for KDA

For KDA, the spatial component uses the same one-step transported-read definition as Qwen, evaluated in KDA's native state coordinates. Write $D_t = \operatorname{diag}(d_{t,1}, \ldots, d_{t,d_k})$. Since $A_t =$

$(I - \beta_t k_t k_t^\top) D_t$, the coordinate read sensitivity is

$$g_{t,i} = (A_t^\top q_t)_i = d_{t,i} \left[q_{t,i} - \beta_t k_{t,i} (k_t^\top q_t) \right], \qquad \omega_i^{\mathrm{KDA}} = \mathbb{E}_{\mathrm{cal}}[g_{t,i}^2]. \tag{20}$$

We normalize this profile in the same way as for Qwen:

$$w_i = \frac{(\omega_i^{\mathrm{KDA}})^{\gamma/2}}{\mathrm{GM}_j\big((\omega_j^{\mathrm{KDA}})^{\gamma/2}\big)}, \qquad \gamma = 0.25. \tag{21}$$

The resulting factors enter the same row-scale rule $r_i = m_i^{1/2} w_i^{-1/2}$ and weighted column fit used for Qwen. The column-fitting objective weights squared reconstruction errors by w_i^2. KDA's post-convolution keys, queries, and gates are measured in native coordinates.

Code normalization and shared fitting. For KDA, $\mathcal{C}_b = \{2^{8-b} q : q \in \mathbb{Z},\ |q| \le 2^{b-1} - 1\}$. Only the b-bit integer q_{ij} is stored; reconstruction and fitting use $z_{ij} = 2^{8-b_i} q_{ij}$. Qwen's 4/6/8-bit codebooks use the same integer bounds without this fixed normalization. Let $\mathcal{I}$ denote the non-pivot integer rows in a head. Writing $v_{ij} = r_i z_{ij}$, the fixed-code update is $c_j = \sum_{i \in \mathcal{I}} w_i^2 v_{ij} X_{ij} / \sum_{i \in \mathcal{I}} w_i^2 v_{ij}^2$, with a positive numerical floor. A zero denominator retains the previous scale.

C.2 Calibration Data, Precision Candidates, and FP16 Pivots

Both models, at both precision budgets and with both weight formats, use 32 WikiText-2 training segments of 2,048 tokens each.

Table C1: **Model-specific STEPQuant configuration.** FP16 pivot and scale-metadata costs are accounted for separately from the nominal bit budget.

Property	Qwen3.8-27B	Kimi-Linear-48B-A3B
Recurrent layers	48	20
State heads per recurrent layer	48	32
Allocation unit	entire head	key row
Number of allocation units	2,304	81,920
Integer candidates (@4 / @6)	$\{2,4,6,8\}$ / $\{4,6,8\}$	$\{2,4,6,8\}$ / $\{4,6,8\}$
Optimizer	multiple-choice DP	Lagrangian allocation
FP16 pivots	32 heads	512 key rows
Pivot ranking	residual allocation risk	distortion-reduction risk

Qwen calibration. The row-impact-weighted MSE in Equation 17 supplies the head-level distortion term for Lifetime-aware Bit Allocation. The spatial component fits and writes the serving state in native coordinates.

KDA calibration. KDA combines gate-derived lifetime with the per-row candidate-format MSE in Equation 18. Its calibrated row-impact scores supply the spatial component's preconditioning factors. Pivot selection scores the lifetime-weighted reduction from integer reconstruction error to FP16 reconstruction error. With pivots fixed, candidate distortions are calibrated excluding those rows from the shared integer column fit, and remaining rows are allocated under the integer budget. Candidate distortions are frozen for discrete allocation. Online Key-Row-Aware Dual-axis Fitting subsequently refits the shared scales to each updated state.

Allocation optimizers. Qwen uses a multiple-choice dynamic program over candidate precisions and remaining budget. KDA minimizes the Lagrangian per unit for a shared budget multiplier, then repairs the discrete assignment to meet the integer budget. The precision menu and budget change between four-bit and six-bit settings, while the lifetime-weighted objective remains the same.

C.3 Recurrent Update and Quantized Writeback

The codebooks and fixed-code column-scale update are specified in Appendix C.1. The steps below combine this fitting with the recurrent update and packed-state storage.

Table C2: **One persistent STEPQuant decode step.** The same logical workflow supports four-bit and six-bit budgets.

Inputs: compressed state. $q_t, k_t, v_t, D_t, \beta_t$. Frozen precision map, pivot mask, and row-impact scores.
1. Reconstruct the previous state from integer codes and scales, or from FP16 values for pivot units.
2. Compute $X = D_t\widehat{S}_{t-1} + \beta_t k_t(v_t^\top - k_t^\top D_t\widehat{S}_{t-1})$.
3. Emit $\widehat{y}_t = X^\top q_t$ before requantization and continue subsequent model computation.
4. Derive row factors using Key-Row-Aware Dual-axis Fitting (Section 4.3). Qwen's two-bit path shares row factors within value groups.
5. For KDA, jointly fit one shared column-scale vector with squared row-impact factors over all non-pivot integer rows within each head. Qwen fits each head at its assigned precision. Its two-bit path uses signed magnitude levels.
6. Pack integer codes and scales. Store pivot values in FP16. Keep the old representation valid until its readers finish.
7. Complete writeback before the next recurrent step uses the new page.

Persistent output: updated packed codes, scales, and FP16 pivots.

D Storage Cost of Codes, Scales, and FP16 Pivots

Let $N = L_h n_h d_k d_v$ denote all recurrent-state elements of one request. Qwen has $N = 48 \cdot 48 \cdot 128^2 = 37,748,736$. KDA has $N = 20 \cdot 32 \cdot 128^2 = 10,485,760$. The FP32 recurrent-state representation is $4N$ bytes. Full-attention KV caches, convolution state, model weights, shared metadata, temporary workspaces, and allocator overhead are separate from this representation.

Compact dual-axis representation. For a head storing one FP16 row vector and one FP16 column vector, these two vectors contribute

$$\frac{16(d_k + d_v)}{d_k d_v} = 0.25 \quad \text{bits per element,} \qquad d_k = d_v = 128.$$

In Qwen's six-bit configuration, 32 pivots replace eight-bit heads with FP16. Including two scale vectors for every head gives

$$b_{\text{Qwen}} = 6 + \frac{16}{128} + \frac{16}{128} + \frac{32}{2304}(16 - 8) = 6.3611. \tag{22}$$

This analytical count reserves two scale slots for every head, including pivots; the packed serving layout omits pivot scale slots.

For the Qwen four-bit configuration, the evaluated shared-scale recurrent-state representation contains 21,804,032 bytes per request, including FP16 scales and 32 FP16 pivot heads. Its compact cost is 4.6209 bits/value, giving a $150,994,944/21,804,032 = 6.9251$ recurrent-state representation ratio relative to FP32. Figure 1 rounds this ratio to $6.93\times$. Compact counts exclude tensor-parallel padding and allocator overhead.

KDA uses row-level allocation with one shared FP16 column-scale vector per head at both budgets. Codebook normalization is fixed by precision and consumes no metadata. Thus one row vector and one column vector cost $16/128 + 16/128 = 0.25$ bits/value, independent of the number of active precisions (three at six bits, up to four at four bits). For reporting, we use the nominal pre-pivot code budget $\bar{b}$ and count 512 FP16 pivot rows as replacements for eight-bit rows at both budgets. Thus the residual integer budget is $\bar{b}N - 8N_{\text{piv}}$, where $N_{\text{piv}} = 512d_v$. Under this accounting convention, the compact count is

$$b_{\text{KDA}}(\bar{b}) = \bar{b} + \frac{16}{128} + \frac{16}{128} + \frac{512}{81920}(16 - 8) = \bar{b} + 0.30. \tag{23}$$

This gives 4.30 and 6.30 bits/value at the four- and six-bit budgets, respectively, for both weight formats. The corresponding per-request sizes are 5.375 and 7.875 MiB, versus 40 MiB for FP32, giving $7.44\times$ and $5.08\times$ compression. These are analytical compact-format counts at the nominal budget, rather than measured allocation sizes. Tensor-parallel page padding and allocator overhead are excluded.

Table D3: **Compact recurrent-state representation (bits/value).** Integer codes, FP16 scales, and pivot replacement are included. Qwen@4 uses the packed byte count above; the other entries follow Equations 22 and 23.

Model	Weights	STEPQuant@6bit	STEPQuant@4bit
Qwen	BF16	6.361	4.621
Qwen	W4A16	6.361	4.621
KDA	BF16	6.300	4.300
KDA	W4A16	6.300	4.300

E Per-Task Accuracy and Generation Length

E.1 Scoring Generated Answers on Short Tasks

The six short tasks use greedy generation and score the answer extracted from the generated response, rather than ranking candidate likelihoods. A response that does not provide an extractable answer is counted as incorrect. WinoGrande has two choices, yet uniform INT4 scores 46.57% on Qwen and 17.36% on KDA (Table 2). At this precision, some generations lose the requested answer format and do not supply a valid choice. Thus accuracy can fall below 50% even on a binary task: the score also captures the model's ability to follow the instruction and produce an answer during decode.

E.2 Generation length with BF16 weights

Table E4 reports the task-level generation lengths underlying Figure 3a–b. Each entry averages all evaluated outputs for that task, including thinking, incorrect answers, and capped outputs. The final column averages the seven task means before rounding.

Table E4: **Mean generated tokens with BF16 weights (thousands).**

State	LCB v6	EvalPlus	AIME 26	MATH-500	HMMT	GPQA-D	IFBench	Avg.
Qwen3.8-27B								
FP32	5.64	0.62	9.73	1.67	18.15	5.10	4.71	6.52
INT8	9.74	0.87	15.44	2.32	29.95	6.04	9.02	10.48
INT6	19.63	1.43	25.62	5.68	33.24	11.31	16.72	16.23
INT4	29.51	2.17	53.28	36.30	50.74	48.62	53.53	39.16
STEPQuant@6bit	**4.44**	**0.64**	**8.51**	**1.62**	**20.11**	**5.12**	**5.89**	**6.62**
STEPQuant@4bit	**7.46**	**0.69**	**13.01**	**1.65**	**20.55**	**4.95**	**7.39**	**7.96**
Kimi-Linear-48B-A3B-Instruct								
FP32	9.47	1.10	18.89	4.17	33.64	9.15	6.70	11.87
INT8	12.86	1.94	27.55	4.20	38.86	12.41	6.10	14.84
INT6	13.75	3.35	29.29	7.47	40.39	17.36	10.45	17.44
INT4	22.32	15.86	63.40	37.73	64.22	57.12	26.84	41.07
STEPQuant@6bit	**9.95**	**1.25**	**20.11**	**3.92**	**31.73**	**8.64**	**6.25**	**11.70**
STEPQuant@4bit	**9.80**	**1.27**	**21.34**	**3.14**	**32.32**	**7.31**	**5.61**	**11.54**

E.3 Generation length with W4A16 weights

At the four-bit budget with W4A16 weights, Qwen's mean accuracy is 0.34 percentage points below FP32, while mean generation length rises from 7.22K to 9.50K tokens (approximately 31.6%). KDA's mean accuracy falls by 2.29 points, with mean length increasing from 12.70K to 13.26K (approximately 4.4%).

Table E5: **Mean generated tokens with W4A16 weights.** Values are thousands of output tokens over all evaluated samples, with the same definition and task ordering as Table E4.

State	LCB v6	EvalPlus	AIME 26	MATH-500	HMMT	GPQA-D	IFBench	Avg.
Qwen3.8-27B								
FP32	8.99	0.91	10.87	1.66	18.00	4.97	5.11	7.22
STEPQuant@6bit	**8.77**	**0.89**	**11.03**	**1.67**	**21.37**	**5.02**	**5.19**	**7.70**
STEPQuant@4bit	**10.29**	**0.95**	**13.56**	**1.80**	**23.88**	**5.25**	**10.80**	**9.50**
Kimi-Linear-48B-A3B-Instruct								
FP32	9.27	1.35	24.62	4.07	34.20	8.92	6.49	12.70
STEPQuant@6bit	**9.36**	**1.38**	**23.58**	**4.00**	**34.65**	**8.99**	**6.40**	**12.62**
STEPQuant@4bit	**10.30**	**1.41**	**25.67**	**4.37**	**35.80**	**9.11**	**6.19**	**13.26**

E.4 Additional Component Ablation

As a supplement to the component ablation in Sec. 5.4, Table E6 reports the corresponding results on Kimi under nominal 4- and 6-bit budgets. Consistent with Qwen, spatial fitting outperforms DSQ, sparse FP16 pivots improve the temporal component, and combining both components achieves further accuracy gains.

Table E6: Component ablation on Kimi-Linear-48B-A3B-Instruct. Accuracy (%) with BF16 weights. Avg. denotes the average across three long-generation benchmarks.

Nominal 6-bit budget

Variant	AIME	GPQA	LCB	Avg.
FP32	68.33	69.70	54.52	64.18
INT6	22.71	57.83	42.37	40.97
Q-Mamba@6bit	38.54	57.07	44.55	46.72
Spatial only	63.54	66.67	52.80	61.00
Temporal w/o pivots	41.46	59.85	48.87	50.06
Temporal only	61.67	66.16	48.91	58.91
STEPQuant@6bit	**67.71**	**68.43**	**54.86**	**63.67**

Nominal 4-bit budget

Variant	AIME	GPQA	LCB	Avg.
FP32	68.33	69.70	54.52	64.18
INT4	0.00	14.58	28.82	14.47
Q-Mamba@4bit	0.00	10.10	28.91	13.00
Spatial only	26.67	50.51	39.51	38.89
Temporal w/o pivots	4.58	35.35	33.74	24.56
Temporal only	8.33	40.91	38.01	29.08
STEPQuant@4bit	**59.17**	**64.14**	**53.29**	**58.87**

F SGLang Integration, Decode Throughput, and State-Pool Memory

F.1 Packed-State Integration in SGLang

We implement the compressed-state path in SGLang (Zheng et al., 2024), pinned to version 0.5.12, with checkpoint-specific precision maps, pivot identities, preconditioners, and packed-state kernels. The adapter maps request slots and tensor-parallel shards onto the shared recurrent-state representation. Prefill unpacks active states, runs the native chunk kernel, and packs the resulting boundary states. Decode updates integer pages directly using floating-point tiles, with no persistent full-matrix FP32 shadow in the serving path. Slot initialization and reuse follow SGLang's request lifecycle. As described in Section 4.4, reconstruction, the Delta update, and the current readout are fused. After the readout, scale fitting and packed writeback run on a separate CUDA stream while later layers process the token. Writeback completes before the next recurrent step reads the state. Benchmark evaluations use the SGLang serving path.

For Qwen STEPQuant@6bit, the packed recurrent-state pages occupy 29,999,104 bytes (28.609 MiB) per request, including integer codes, FP16 scales, and FP16 pivots, compared with 150,994,944 bytes (144 MiB) for FP32. This gives a $5.03\times$ storage reduction (80.13%).

F.2 Decode Timing Protocol and Throughput Results

We measure decode throughput from consecutive CUDA start events on each TP rank, including gaps between decode steps. For each run, elapsed decode time is the sum of those intervals on the slowest rank. Dividing the number of request-token transitions by this time gives tokens/s. Prefill,

startup, warmup, and final output delivery lie outside this interval. Each configuration has three runs after one warmup run. We report the median throughput.

All batches use four A800 GPUs with TP4 and the same 128-token prompt for every request, with 1,024 generated tokens. Decoding is greedy. Every recorded decode step retains its stated batch size. FP32 state and STEPQuant@6bit share server settings within each pair. The compressed path includes fitting, FP16 pivots, and writeback.

F.3 Memory Reserved for Concurrent Requests

Let B denote supported concurrent requests, L context length, W shared text weights, S one aggregate persistent-state slot, and K attention KV per token. With reservation ratio r and one additional sentinel slot,

$$M(B, L) = W + (rB + 1)S + BLK. \tag{24}$$

For Qwen, an FP32 slot contains 150,994,944 bytes (144 MiB) of recurrent matrices and 2,949,120 bytes (2.8125 MiB) of convolution state, so $S = 146.8125$ MiB. Across 16 full-attention layers, $K = 65,536$ bytes (64 KiB) per context token.

For the radix-caching configuration considered here, SGLang reserves three slots per supported request. Overlap tracking adds two, plus one global sentinel slot (Zheng et al., 2024). Figure 1a uses this five-slot configuration, $(5B + 1)S$, before and after compression. Under this five-slot reservation, the state term is independent of context length.

Weights are counted from the local checkpoints' safetensors payloads, including integer codes, scales, zero points, and retained BF16 text weights in the W4A16 model. Vision, MTP draft, and shape metadata are excluded. The exact text-weight payloads are 53,791,996,928 bytes (BF16) and 17,776,688,128 bytes (W4A16). All counts are summed across tensor-parallel ranks for one model replica. The analytical capacity curve reserves 30,015,488 bytes of compact recurrent-state representation, including pivot scale slots, plus the unchanged 2,949,120-byte convolution state, totaling 31.4375 MiB. Applying this format uniformly to the reserved slots gives 9.85 GiB of reserved persistent-state pool capacity at $B = 64$, compared with 46.02 GiB in FP32. Weights and attention KV are excluded from the state curves. Horizontal lines show the two weight payloads.

F.4 Additional Results on KDA

Section 5.6 presents the serving-level and state-level efficiency results on Qwen. Here, we provide the corresponding results on KDA under the same evaluation settings.

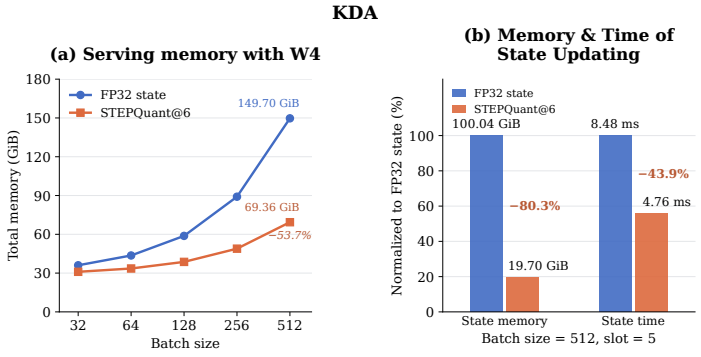

Figure F4: Serving memory and state-update time of STEPQuant on KDA.

As shown in Fig. F4, STEPQuant@6bit reduces total serving memory from 149.70 to 69.36 GiB (53.7%) at a batch size of 512. At the state level, memory consumption decreases from 100.04 to 19.70 GiB (80.3%, 5.08× compression), while state-update time decreases from 8.48 to 4.76 ms (43.9%, 1.78× faster). These results further demonstrate the memory and computational benefits of STEPQuant on KDA.

F.5 Decode Throughput

We evaluate full-model decode throughput with BF16 weights on four NVIDIA A800 GPUs using tensor parallelism of four (TP4). We test batch sizes $B \in \{32, 64, 128, 256, 512\}$, with 128 input tokens and 1,024 generated tokens per request. The measurements include inter-step gaps, quantization fitting, and compressed-state writeback.

Table F7: **Decode throughput.** Units are tokens/s. Gains use the unrounded throughputs.

Model	Batch	FP32	STEPQuant@6bit	Gain
Qwen	32	1,656	1,727	+4.32%
Qwen	64	2,980	3,122	+4.77%
Qwen	128	4,157	4,485	+7.87%
Qwen	256	5,655	6,360	+12.47%
Qwen	512	6,040	7,280	+20.53%
KDA	32	5,248	5,356	+2.06%
KDA	64	8,284	8,523	+2.89%
KDA	128	13,284	13,936	+4.90%
KDA	256	16,979	18,505	+8.99%
KDA	512	21,241	23,748	+11.80%

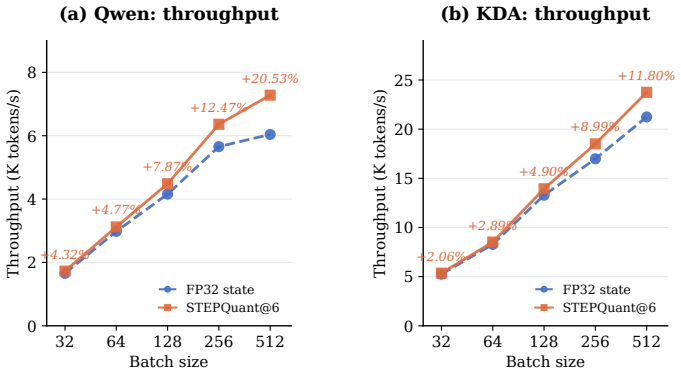

Figure F5: Full-model decode throughput.

As shown in Fig. F5, STEPQuant@6bit consistently improves decode throughput on both models, with larger gains at higher batch sizes. At $B = 512$, throughput increases by 20.53% on Qwen (from 6,040 to 7,280 tokens/s) and 11.80% on KDA (from 21,241 to 23,748 tokens/s). Detailed results are reported in Table F7.

G Comparison with DAMP

The concurrent work DAMP (Zhang et al., 2026) uses reconstruction error and decay-based persistence to select FP16 key channels, while quantizing the remaining channels to INT8 with a Hadamard transform. In contrast, STEPQuant combines lifetime-aware mixed-precision allocation with key-row-aware dual-axis fitting, accounting for both error persistence and readout impact. This enables STEPQuant to achieve near-FP32 accuracy at substantially lower effective precision in our evaluations.

Since DAMP's implementation was unavailable for reproduction and the two studies use different generation settings, we compare relative FP32 accuracy on three shared KDA benchmarks. We directly extract DAMP and its INT8, INT4, and NVFP4 baseline results from the original paper and normalize each method's three-task average by its corresponding FP32 baseline.

Table G8: Accuracy retention on three shared KDA benchmarks: AIME 2026, HMMT Feb 2026, and LiveCodeBench v6. Retention is normalized to each study's FP32 baseline.

Method	Avg. bits	Retention (%)
DAMP	9.9	100.99
STEPQuant@6bit	6.3	100.51
STEPQuant@4bit	4.3	92.91
INT8†	9.0	83.54
INT4†	5.0	6.66
NVFP4†	4.5	6.74

† Baseline results reported by DAMP.

As shown in Table G8, DAMP retains 100.99% of FP32 accuracy at 9.9 effective bits per state value, while STEPQuant achieves 100.51% at 6.30 bits and 92.91% at 4.30 bits. In contrast, the INT8, INT4, and NVFP4 baselines reported by DAMP retain 83.54%, 6.66%, and 6.74%, respectively. These results demonstrate STEPQuant's ability to preserve near-FP32 accuracy at low precision (6.30 vs. 9.9 bits), although differences in evaluation settings preclude a strictly controlled cross-study comparison.

H LIMITATIONS

STEPQuant's lifetime weight approximates error persistence through gate decay without fully modeling the time-varying, key-dependent state transition, and therefore does not fully capture the long-term effects of quantization error. With BF16 weights at four bits, KDA loses accuracy on long-generation tasks, while Qwen generates longer outputs despite retaining near-FP32 average accuracy. Our evaluation covers two GDN/KDA models under fixed hardware and workload settings; accuracy and systems gains on other architectures and under dynamic serving workloads remain to be verified.