FACTORSTATE: DYNAMIC LOW-RANK TRACKING FOR MEMORY-EFFICIENT GATED DELTA NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context inference incurs growing memory costs from attention key–value (KV) caches. Gated Delta Networks (GDNs) mitigate this growth by summarizing history in fixed-size recurrent matrix states rather than maintaining a growing KV cache. Yet in hybrid large language models, these states are replicated across layers, heads, and concurrent requests, leaving a substantial aggregate memory footprint. Existing work has revealed low-rank structure in recurrent states and used it to motivate compression, but rank alone does not determine whether retained subspaces remain useful during decoding. We identify a temporal phenomenon, *capacity–geometry decoupling*. At a fixed rank, optimal approximations of the current state maintain nearly unchanged normalized squared reconstruction error, while reusing earlier subspaces leads to growing reconstruction and memory-read errors. Motivated by this observation, we introduce FACTORSTATE, a training-free representation that tracks the evolving recurrent state with compact factors under a fixed memory budget. FACTORSTATE performs recurrent reads and updates directly in factor space and periodically recompresses the factors, leaving pretrained weights and dense prompt processing unchanged. Experiments across Qwen and OLMo models show that FACTORSTATE reduces recurrent-state storage by approximately 61% while achieving task performance close to dense execution. GPU evaluations further demonstrate lower execution time and peak memory under both fixed-active-batch and continuous-batching workloads.

1 INTRODUCTION

Long-context inference increases the memory requirements of large language models (LLMs). Attention layers retain key–value (KV) pairs from previous tokens, creating caches that grow with sequence length for every active request. Hybrid architectures reduce this cost by combining attention with recurrent layers that summarize history in fixed-size states. Gated Delta Networks (GDNs), in particular, combine gated forgetting with targeted memory updates in a recurrent matrix, supporting adaptive memory management without increasing the size of the recurrent state at each step (Yang et al., 2025). These properties make GDNs promising for efficient long-context inference.

The total memory footprint of recurrent states nevertheless remains substantial in hybrid LLMs. Each active request maintains separate matrices across recurrent layers and heads, and decoding repeatedly reads and updates these matrices. Reducing their storage therefore offers an opportunity to improve inference efficiency. Existing work provides a starting point. Distinct low-rank and high-rank heads have relatively stable rank profiles, which have been used to guide pruning (Sun et al., 2026). Structural pruning of query and key channels has also been used to reduce state size (Nazari & Rusch, 2026). These studies establish opportunities for reducing state storage, but rank alone does not determine whether subspaces selected earlier remain useful as decoding proceeds.

Our analysis of dense state trajectories reveals a temporal phenomenon that we call *capacity–geometry decoupling*. As shown in Figure 1, recomputing optimal subspaces from each evaluated state at a fixed rank yields nearly unchanged normalized squared reconstruction error, while reusing earlier subspaces leads to growing reconstruction and memory-read errors. Rank specifies how many directions are retained. Geometry specifies which directions they are. Both approximations are evaluated on the same dense states with the same rank budget. This comparison separates the error imposed by limited capacity from the additional error caused by reusing earlier subspaces, without confounding the result

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.

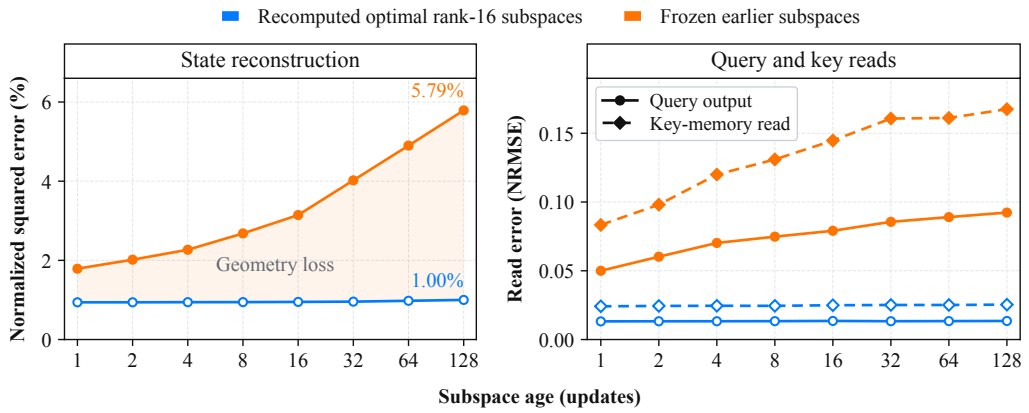

Figure 1: **Low-rank capacity remains sufficient while frozen subspaces become stale.** Both panels use the same 16 Qwen3.5-4B teacher-forced streams at rank 16. (Left) At each evaluated time, C is computed from optimal rank-16 subspaces of the current dense state and stays near 1%, while the residual $C + G$ from reusing earlier subspaces increases. Shading denotes the extra geometry loss G, not uncertainty. (Right) The normalized root-mean-square error (NRMSE) against corresponding dense reads increases for frozen subspaces in both post-update query outputs (solid circles) and decayed pre-update key-memory reads (dashed diamonds). The blue read curves use the recomputed rank-16 approximation of each branch's state. Lines connect measured points on a base-2 logarithmic age axis. Appendix A.3 reports confidence intervals and lower bounds for age-128 minus age-1 contrasts, using 10,000 category-stratified whole-stream bootstrap resamples. Inputs are fixed throughout this comparison. Free-running tasks are evaluated separately.

with differences in generated tokens. We observe this separation across Qwen and OLMo trajectories, with growing errors in both the key-based reads used to update the state and the query-based reads used to produce outputs. Thus, fixed subspaces can lose accuracy even when the rank remains sufficient, motivating a representation that tracks the changing state geometry.

Motivated by this observation, we introduce FACTORSTATE, a training-free representation that maintains the evolving GDN state as compact factors under a fixed memory budget. After a single factorization following prompt processing and before decoding, FACTORSTATE reads and updates the state directly through these factors. Each write adds one column to each factor. Periodic recompression updates the retained subspaces and restores the target rank. Model weights and dense prompt processing remain unchanged. Counting buffered writes and the current write slot, the default configuration uses approximately 61% less recurrent-state storage than dense execution (Dense). Both methods use FP32 for recurrent-state storage and BF16 for model weights in our CUDA evaluations. When replaying the same inputs recorded from dense execution, FACTORSTATE achieves lower state and read errors than the tested Static, Oja, and COD baselines at nearly equal state-storage cost. Task evaluations across Qwen and OLMo yield scores close to dense execution, while our CUDA implementation reduces execution time and peak memory on the evaluated workloads with fixed active batches and continuous batching.

We summarize our contributions as follows:

- We identify and characterize *capacity–geometry decoupling*, the temporal separation between low-rank capacity and subspace aging, and measure its effects on both GDN memory reads across multiple ranks and model families.
- We develop FACTORSTATE, a training-free dynamic factor representation that tracks the current recurrent matrix and applies the GDN update directly to its factors, with buffered writes and periodic recompression to bound persistent storage.
- We demonstrate approximately 61% recurrent-state storage savings with scores close to Dense on text and image–text tasks, alongside faster execution and lower peak GPU memory on the evaluated workloads.

2 CAPACITY–GEOMETRY DECOUPLING IN GATED-DELTA MEMORY

2.1 GATED-DELTA RECURRENT MEMORY

Consider token index t in one normalized Gated Delta Network (GDN) head (Yang et al., 2025). Let $S_{t-1} \in \mathbb{R}^{d_k \times d_v}$ denote the recurrent matrix retained after processing token $t-1$, where d_k and d_v are the key and value dimensions. The head receives an ℓ_2-normalized query $q_t \in \mathbb{R}^{d_k}$, an ℓ_2-normalized key $k_t \in \mathbb{R}^{d_k}$, a value $v_t \in \mathbb{R}^{d_v}$, a log-decay $g_t \in \mathbb{R}$, and a post-activation write coefficient $\beta_t \in \mathbb{R}$. The tested adapters obtain β_t from the model's sigmoid gate, including the model-defined rescaling when present. The dense update is

$$\begin{aligned} \alpha_t &= \exp(g_t), & \bar{S}_t &= \alpha_t S_{t-1}, \\ m_t &= \bar{S}_t^\top k_t, & \delta_t &= \beta_t(v_t - m_t), \\ S_t &= \bar{S}_t + k_t \delta_t^\top, & o_t &= S_t^\top (q_t/\sqrt{d_k}). \end{aligned} \tag{1}$$

The scalar α_t decays the old memory to $\bar{S}_t$. The key read $m_t \in \mathbb{R}^{d_v}$ retrieves the value associated with k_t, and the residual $\delta_t \in \mathbb{R}^{d_v}$ corrects it through a rank-one write. The query then reads the updated state to produce $o_t \in \mathbb{R}^{d_v}$. Dense execution stores all $d_k d_v$ entries of S_t for every active request, head, and recurrent layer.

2.2 MEASURING CAPACITY AND GEOMETRY

The recurrent update can change which subspaces best represent the state. At a fixed rank, we distinguish the error of the optimal approximation to the current state from the additional error incurred by retaining earlier subspaces. We call these capacity loss and geometry loss, respectively.

Let S_t be the dense state after recurrent update t, and let $S_{t,r}$ be its optimal rank-r approximation, recomputed by truncated singular value decomposition (SVD) at each evaluated time. This decomposition is a diagnostic reference, not FactorState's online update. At an earlier anchor update a, let $U_{a,r}$ and $V_{a,r}$ contain the leading r left and right singular vectors. Their two-sided frozen projection is $P_{a,r}(X) = U_{a,r}U_{a,r}^\top X V_{a,r}V_{a,r}^\top$. We compare the rank-$r$ reconstruction loss with the excess loss caused by reusing that anchor geometry:

$$C_{t,r} = \frac{\|S_t - S_{t,r}\|_F^2}{\|S_t\|_F^2}, \qquad G_{a,t,r} = \frac{\|S_t - P_{a,r}(S_t)\|_F^2}{\|S_t\|_F^2} - C_{t,r}. \tag{2}$$

Here $\|\cdot\|_F$ is the Frobenius norm. The capacity loss $C_{t,r}$ measures what the recomputed rank-r approximation leaves out. For fixed orthonormal anchor bases, this diagnostic projection minimizes the Frobenius reconstruction error over all coefficient matrices. Thus, the geometry loss $G_{a,t,r}$ measures the excess error imposed by frozen subspaces, even with optimal coefficients. Both approximations have rank at most r, so their error gap does not come from giving the current approximation more capacity.

Reconstruction error alone does not show whether stale geometry affects the recurrence. We therefore evaluate query reads after writing and key reads after decay but before writing. For branch $b \in \{q, k\}$, the state–input pair and its exact and frozen-geometry reads are

$$\begin{aligned} (X_t^{(q)}, z_t^{(q)}) &= (S_t, q_t/\sqrt{d_k}), & (X_t^{(k)}, z_t^{(k)}) &= (\bar{S}_t, k_t), \\ y_t^{(b)} &= X_t^{(b)\top} z_t^{(b)}, & y_{a,t,r}^{(b,\mathrm{fr})} &= P_{a,r}^{(b)} \Big(X_t^{(b)} \Big)^\top z_t^{(b)}. \end{aligned} \tag{3}$$

Here $\bar{S}_t$ is the exactly decayed pre-update state from Equation 1, and $P_{a,r}^{(b)}$ uses the leading rank-r singular subspaces of the corresponding anchor state $X_a^{(b)}$. We summarize the difference between the two reads with an energy-pooled normalized root-mean-square error (NRMSE). The blue read curves in Figure 1 use the recomputed rank-r approximation of $X_t^{(b)}$. Appendix A.2 gives the pooling and stream-level bootstrap procedures.

2.3 EMPIRICAL SIGNATURE AND DESIGN IMPLICATION

Figure 1 shows capacity–geometry decoupling on Qwen3.5-4B. At rank 16, recomputing the current subspaces keeps the normalized squared reconstruction error near 1%, whereas reusing earlier subspaces leads to growing reconstruction errors and errors in both query and key reads. Multi-rank experiments reproduce this qualitative separation in Qwen and OLMo Hybrid where the current approximation remains accurate. Appendix A.3 reports the full statistics, including the capacity-limited OLMo rank-8 result.

These observations show that sufficient rank capacity alone does not ensure that earlier subspaces remain accurate as the state evolves. They motivate a compact representation whose subspaces can adapt to the current recurrent matrix during decoding. Section 3 develops FACTORSTATE to realize this principle under a fixed memory budget.

3 FACTORSTATE: TRACKING EVOLVING STATE GEOMETRY

FACTORSTATE stores the state retained for the next token (the committed state) as compact dynamic factors. It applies the recurrence in Equation 1 directly to these factors during decoding. Each selected recurrent state is converted once after dense prompt processing. Model weights and attention KV caches remain unchanged. Appendix B.2 describes state conversion and model-specific adapters.

3.1 EXACT RECURRENCE IN FACTOR SPACE

FACTORSTATE stores the committed state as $\hat{S}_{t-1} = U_{t-1}V_{t-1}^\top$, with left factors $U_{t-1} \in \mathbb{R}^{d_k \times c_{t-1}}$, right factors $V_{t-1} \in \mathbb{R}^{d_v \times c_{t-1}}$, and live column count c_{t-1}. A token is processed without materializing a $d_k \times d_v$ matrix:

$$\begin{aligned} \bar{U}_t &= \sqrt{\alpha_t}\, U_{t-1}, \qquad \bar{V}_t = \sqrt{\alpha_t}\, V_{t-1}, \\ \hat{m}_t &= \bar{V}_t(\bar{U}_t^\top k_t), \qquad \hat{\delta}_t = \beta_t(v_t - \hat{m}_t), \\ U'_t &= [\bar{U}_t, k_t], \qquad V'_t = [\bar{V}_t, \hat{\delta}_t], \qquad \hat{o}_t = V'_t(U'^\top_t q_t/\sqrt{d_k}). \end{aligned} \tag{4}$$

Here $\bar{U}_t$ and $\bar{V}_t$ are the decayed factors, and $\hat{m}_t$ and $\hat{\delta}_t$ are the key-memory read and write residual obtained from the represented state. Brackets denote column concatenation, so U'_t and V'_t are the factors after appending the current write, from which the current-token output $\hat{o}_t$ is computed. Balanced decay gives $\bar{U}_t\bar{V}_t^\top = \alpha_t\hat{S}_{t-1}$, and appending the write gives $U'_tV'^\top_t = \alpha_t\hat{S}_{t-1} + k_t\hat{\delta}_t^\top$. These identities reproduce the dense recurrence on the represented state $\hat{S}_{t-1}$. Between recompressions, the factor operations are therefore exact in real arithmetic. Rank truncation changes the state passed to later tokens. Hats distinguish the represented state and its reads from the dense trajectory.

3.2 BUFFERED WRITES AND PERIODIC RECOMPRESSION

Factor columns grow by one after each write. Let r be the persistent target rank, B the number of buffered write columns, and $\mathcal{R}_r$ the rank-r recompression operator. After computing $\hat{o}_t$, the method commits the next state as

$$(U_t, V_t) = \begin{cases} (U'_t, V'_t), & c_{t-1} + 1 < r + B + 1, \\ \mathcal{R}_r(U'_t, V'_t), & c_{t-1} + 1 = r + B + 1. \end{cases} \tag{5}$$

The current output is computed from the expanded factors before recompression. The newly truncated factors first affect token $t + 1$, which receives the committed pair (U_t, V_t). In Figure 2, the buffer count $b_t = c_{t-1} - r$ is measured before the current write. The illustrated r-wide read corresponds to $b_t = 0$, with width c_{t-1} when writes are buffered.

The reference recompression evaluates the rank-r truncation entirely through factor coordinates. For expanded factors U' and V', it computes

$$\begin{aligned} U' = Q_U R_U, \qquad V' = Q_V R_V, \qquad R_U R_V^\top = P\Sigma W^\top, \\ \mathcal{R}_r(U', V') = \left(Q_U P_{:r}\Sigma_r^{1/2},\ Q_V W_{:r}\Sigma_r^{1/2}\right). \end{aligned} \tag{6}$$

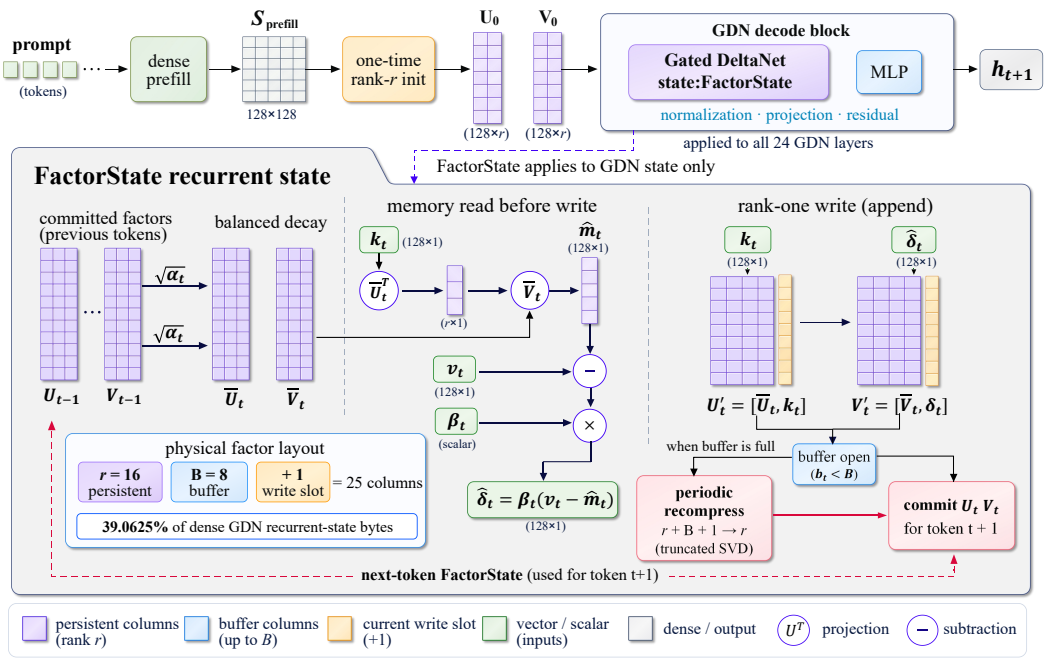

Figure 2: **FactorState initialization and recurrent-state update in Qwen3.5-4B.** After dense prefill, recurrent states in all 24 GDN layers undergo one-time rank-r factorization. The lower panel shows decay, the key-memory read, rank-one writes, and periodic recompression. The query read (not shown) uses the expanded factors before recompression (Equations 4–5). The r16/B8 layout reserves 25 columns, including the current write slot.

Here Q_U and Q_V are the orthonormal bases from the two thin QR factorizations, while R_U and R_V are their upper-triangular factors. The matrices P and W contain the left and right singular vectors of the small core, and Σ is its diagonal singular-value matrix. The subscript $:r$ selects the leading r columns, and Σ_r is the corresponding $r \times r$ diagonal singular-value block. Equation 6 returns balanced rank-r factors without forming the $d_k \times d_v$ recurrent matrix.

Our CUDA implementation uses regularized Cholesky coordinates and Jacobi SVD of a $c \times c$ core (25×25 for r16/B8) to approximate the reference truncation without explicit QR bases. Appendix B.1 describes its three-stage maintenance procedure, used in both quality and performance evaluations.

3.3 Memory and Computational Cost

Memory accounting must include the column whose append triggers recompression. The physical column capacity, number of stored factor scalars, and fraction of dense-state storage are

$$c_{\text{physical}} = r + B + 1, \qquad M_{\text{factor}} = (d_k + d_v)c_{\text{physical}},$$
$$\rho_{\text{state}} = \frac{M_{\text{factor}}}{d_k d_v} = \frac{(d_k + d_v)(r + B + 1)}{d_k d_v}. \tag{7}$$

The physical column capacity c_{physical} comprises r persistent columns, B buffered columns, and one current write. M_{factor} counts factor scalars, and ρ_{state} is their ratio to the $d_k d_v$ dense-state scalars. Our primary rank-16/buffer-8 (r16/B8) point reserves 25 columns. At equal storage dtype, it uses approximately 39.1% of dense GDN recurrent-state bytes for both tested Qwen heads ($d_k = d_v = 128$) and OLMo heads ($d_k = 96, d_v = 192$).

We next consider the corresponding computational cost. With c active factor columns, the factorized read and write require $O((d_k + d_v)c)$ arithmetic. Recompression costs $O((d_k + d_v)c^2 + c^3)$ and is amortized across its update period.

4 EXPERIMENTS

We evaluate state-representation fidelity, full-model task performance, and GPU efficiency, then examine the effects of rank and buffering.

4.1 EXPERIMENTAL SETUP

Models and configuration. Our primary model is Qwen3.5-4B (Qwen Team, 2026a), with additional evaluations on Qwen3.5-9B (Qwen Team, 2026b) and OLMo-Hybrid-Instruct-SFT-7B (Merrill et al., 2026). All models use bfloat16 (BF16) weights. Unless stated otherwise, FACTORSTATE uses rank 16, eight buffered writes (r16/B8), and 32-bit floating-point (FP32) factors with 25 physical columns, including the current write slot. In the CUDA experiments, the Dense Flash Linear Attention (FLA) baseline stores its recurrent states in FP32. We evaluate the Triton implementation across models and CUDA on Qwen3.5-4B. Appendices B.2 and B.1 give model and implementation details.

State-representation evaluation. We study subspace evolution along Dense recurrent-state trajectories. For state comparisons, FP32 reference implementations replay the same queries, keys, values, decay factors, and write gates recorded from Dense execution. Each method computes write residuals from its own approximate state. We compare FACTORSTATE with Static, Oja (Oja, 1982), and Co-Occurring Directions (COD) (Mroueh et al., 2017) at comparable physical storage budgets. All methods are initialized from the same Dense prefill states. Oja hyperparameters are selected on calibration streams and held fixed on disjoint validation streams.

Full-model quality evaluation. Dense and FACTORSTATE independently generate greedy non-thinking responses for IFEval, RULER, GSM8K, MATH-500, EvalPlus, ChartQA, and MathVista, with identical inputs, task-specific output budgets, and scorers within each pair. A separate 400-question text diagnostic compares next-token distributions under shared Dense prefixes, with independent recurrent states. Appendices D.1 and D.5 give the evaluation protocols.

Systems evaluation and statistics. On RTX 3090 and A800, Dense uses optimized FLA GDN kernels. Both methods use FLA convolution. Fixed-active batches use scaled dot-product attention (SDPA) and preallocated KV. Continuous batching uses FlashInfer paged attention and the same GDN CUDA graph policy. Completion time includes admission (prefill, state conversion, and cache setup), decoding, and maintenance. Decode throughput uses synchronized decode wall time. The approximately 61% reduction concerns recurrent-state storage. Peak GPU memory includes conversion and maintenance workspace. State-representation errors use complete replay streams as statistical units, task scores use paired questions, and systems measurements use independent paired runs. Appendices C.2, D.2, and E describe the corresponding statistical procedures.

4.2 STATE-REPRESENTATION FIDELITY

At comparable physical storage, we compare FACTORSTATE with Static's frozen prefill bases, Oja's write-driven basis updates (Oja, 1982), and a GDN adaptation of Co-Occurring Directions (COD) (Mroueh et al., 2017). We use 24 Qwen3.5-4B sequences, six each from natural language, retrieval, mathematics, and code, with 512 prefill tokens and 512 updates. Four FP32 reference implementations replay identical GDN inputs on RTX 3090 (TF32 disabled), each using its own write residuals. Oja calibration on three sequences per category selects learning rate 0.001 from five candidates and period 8 from $\{1, 8\}$. The remaining 12 sequences form the validation set.

FACTORSTATE r16/B8 and COD each reserve 25 paired columns, using 39.06% of Dense FP32 state bytes, while Static and Oja at rank 23 use 39.17%. Figure 3 compares pooled state, query-read, and key-read errors. Appendix C defines the controls, readout timing, storage accounting, and paired confidence intervals.

FACTORSTATE reduces both state and read errors. Relative to COD, the best control on all three pooled metrics, FACTORSTATE reduces final-state error by 59.95%, query-read NRMSE by 69.62%, and key-read NRMSE by 67.40%. All three errors are lower on every validation stream. For each metric, the paired 95% interval for the ratio of pooled errors, FactorState/COD, lies below one.

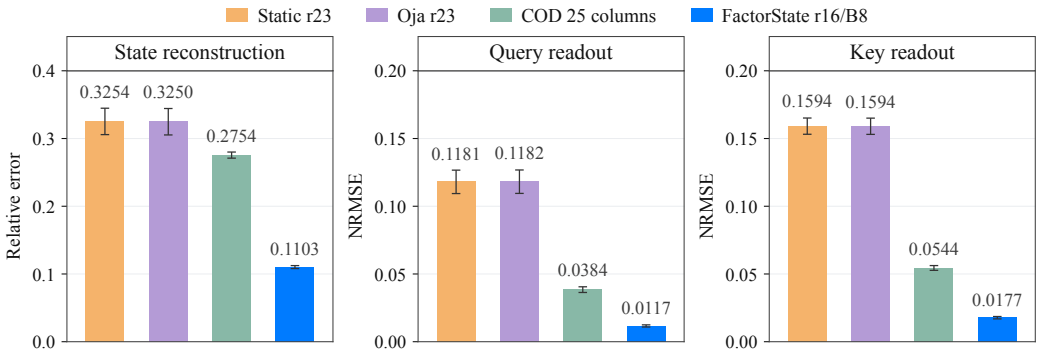

Figure 3: **State-representation fidelity at matched memory.** Four FP32 methods replay identical inputs on 12 validation streams. (Left) State error after maintenance at update 512. (Center) Query-read NRMSE when each method produces its output. (Right) Decayed pre-write key-read NRMSE, without query scaling. We divide summed squared errors by summed Dense reference energies and take the square root. Error bars show 95% whole-stream bootstrap intervals (10,000 resamples). Static/Oja use 39.17% and COD/FactorState 39.06% of Dense FP32 representation bytes. Lower is better. The two read panels share a scale.

Conditional errors grow modestly over 4,096 updates. Across eight teacher-forced Qwen3.5-4B streams, we compute each stream's error at update 4,096 relative to update 1,024. The geometric mean of these ratios is 1.144 for state error (95% confidence interval (CI) $[1.096, 1.190]$) and 1.100 for query-read error (CI $[1.042, 1.156]$), under shared dense inputs. Appendix C.3 gives the protocol, absolute errors, and paired confidence intervals.

4.3 Benchmark Performance

Table 1 reports cross-model benchmark performance at r16/B8 using Triton. We also evaluate the CUDA implementation used in system measurements on seven Qwen3.5-4B benchmarks: score differences from Dense range from -1.80 to $+0.74$ pp (Figure 4). Appendix D.1 details inputs, generation settings, and scoring.

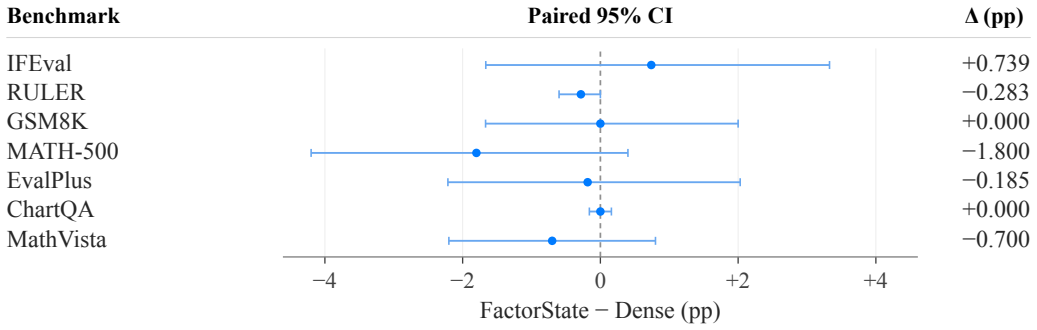

Figure 4: **CUDA benchmark quality on Qwen3.5-4B** (5,683 pairs). Points show FactorState minus matched Dense scores (pp). Bars show paired 95% intervals. FactorState uses the CUDA implementation timed in Section 4.4. Complete scores, math output budgets, and uncertainty are reported in Appendix D.4.

On 400 of the 2,183 CUDA text questions, mean Kullback–Leibler (KL) divergence under Dense-generated prefixes ranges from 0.00105 to 0.00974 nats across four tasks. Top-1 agreement is 96.09–99.63%. Both mathematics datasets use a 2,048-token diagnostic cap. Appendix D.5 gives the sampling and replay protocols and per-task diagnostic results.

Table 1: Cross-model benchmark performance with Triton at r16/B8. T denotes text-only input, and I+T denotes image–text input (multimodal). Scores are percentages. n counts paired cases. Δ is FactorState minus Dense in percentage points, computed from unrounded scores. Appendix D.3 reports confidence intervals.

Model	Benchmark	Input	n	Dense	FactorState	Δ (pp)
Qwen3.5-4B	IFEval strict prompt	T	541	82.810	82.440	-0.370
	RULER (51 cells)	T	2,550	85.556	85.554	-0.001
	GSM8K/MATH-500	T	800	86.000	85.875	-0.125
	EvalPlus	T	542	71.587	72.509	$+0.923$
	ChartQA test	I+T	2,500	84.480	84.400	-0.080
	MathVista testmini	I+T	1,000	78.000	78.800	$+0.800$
Qwen3.5-9B	RULER hard subset	T	500	84.910	84.990	$+0.080$
	GSM8K/MATH-500	T	800	88.875	87.500	-1.375
	EvalPlus	T	542	74.539	74.908	$+0.369$
	ChartQA test	I+T	2,500	87.080	86.960	-0.120
	MathVista testmini	I+T	1,000	78.700	79.600	$+0.900$
OLMo Hybrid 7B	RULER subset	T	180	83.315	83.130	-0.185
	GSM8K/MATH-500	T	800	77.125	76.375	-0.750
	EvalPlus	T	542	66.605	65.867	-0.738

4.4 Memory and Runtime

FactorState reduces complete-request time at the tested batch sizes. Table 2 compares fixed active batches with preallocated KV. Complete-request timing includes admission, decoding, token selection, and maintenance, excluding model loading, tokenization, and independent warmup.

Table 2: Fixed-active-batch performance. C: concurrency. T: decode steps. Values are per-method medians. Δ uses median paired FactorState/Dense ratios. Throughput: total decoded tokens divided by synchronized decode wall time. Memory: peak allocation over the complete request.

Metric	Method / change	RTX 3090 ($T = 512$)	A800 ($T = 2048$)		
		$C = 192$	$C = 192$	$C = 384$	$C = 512$
Request time (s) $\downarrow$	Dense	48.54	98.31	166.50	223.84
	FactorState	37.47	88.66	127.42	169.36
	Δ (%)	-22.81	-9.82	-23.47	-24.33
Decode throughput (tokens/s) $\uparrow$	Dense	2,103.89	4,027.14	4,760.78	4,720.78
	FactorState	2,921.04	4,538.91	6,373.00	6,393.02
	Δ (%)	$+38.86$	$+12.71$	$+33.86$	$+35.40$
Peak allocated memory (GiB) $\downarrow$	Dense	21.61	30.61	52.66	67.35
	FactorState	15.92	24.92	41.21	52.10
	Δ (%)	-26.32	-18.58	-21.74	-22.65

Faster decoding offsets added admission cost in all four configurations in Table 2. At A800 concurrency 192, results are mixed at 512 decode steps, while both paired runs are faster at all measured checkpoints from 768 steps onward, as reported in Appendix E.1.

Continuous batching with mixed output lengths. We evaluate Qwen3.5-4B with 1,024 distinct 64-token prompts on RTX 3090 and 4,096 on A800, allowing at most 96 and 512 outstanding requests, respectively. A new request enters when a completed request releases its slot. Output lengths are 256, 512, 1,024, or 2,048 tokens, with equal counts at each length. Both methods share paged attention and a GDN CUDA graph policy. On RTX 3090, our CUDA implementation reduces the time to complete all requests by 13.83% and increases decode-wall throughput by 21.83%. Peak allocated memory falls from 21.32 to 19.36 GiB. On A800, completion time falls by 15.76%, decode-wall

throughput rises by 29.57%, and peak allocation falls from 68.57 to 54.80 GiB. Appendix E.4 reports the protocols and measurements for each workload.

4.5 Rank and Buffer Ablations

The default r16/B8 setting was selected before these ablations. We compare Dense, r16/B0, r16/B8, and r24/B0 using CUDA on RTX 3090 at concurrency 192 and 512 decode steps. At fixed rank 16, buffering increases physical capacity from 17 to 25 columns and reduces recompression from every update to every nine updates. Request time falls from 50.67 to 36.27 seconds. At equal 25-column storage, r24/B0 takes 73.93 seconds, 2.04 times the r16/B8 request time, with nearly identical peak allocated memory. These comparisons support r16/B8 as a practical memory–runtime operating point among the tested configurations. Full performance results, task-dependent quality trade-offs on a frozen 400-case subset, and paired intervals appear in Appendix E.3.

5 Related Work

Recurrent memory capacity and compression. Fixed-size matrix memories, including Gated Delta Networks (Yang et al., 2025), motivate both capacity expansion and compression. SSE, StateX, and Sparse Delta Memory enlarge recurrent capacity (Pan et al., 2026; Shen et al., 2026; Cabannes et al., 2026). HOLA supplements the state with bounded exact KV memory (Cui, 2026). From the compression perspective, head-wise rank stratification has been characterized and used to guide pruning (Sun et al., 2026). Query/key channels have also been structurally pruned, with recovery fine-tuning evaluated (Nazari & Rusch, 2026). These approaches change the available memory capacity or select which components to retain. Our study of the evolving state of a pretrained GDN shows that an accurate approximation at a fixed rank does not ensure that previously captured subspaces remain useful. This motivates tracking state geometry during inference while keeping the pretrained weights and projections unchanged.

State buffering and deferred materialization. Buffering recent updates can reduce the cost of accessing recurrent state. KVBuffer batches state updates and, for short contexts, reads directly from buffered keys and values without materializing a recurrent state (Zou & Zhong, 2026). ReplaySSM caches inputs around a checkpoint (Liou & Dao, 2026), while DeltaLog retains a dense base with a bounded update log (Lin et al., 2026). In tree speculative decoding, Bole encodes candidate updates as token-level factors and reconstructs only the selected state (Wang et al., 2026). These designs reduce state traffic or transient verification storage. FactorState targets persistent recurrent-state storage by keeping the committed state factorized and periodically recompressing buffered writes.

Online subspace tracking and low-rank updates. Oja's rule estimates principal directions from streaming observations (Oja, 1982). Co-Occurring Directions approximates matrix products through paired factors and singular-value shrinkage (Mroueh et al., 2017), while incremental SVD updates singular subspaces under low-rank modifications (Brand, 2006). FactorState maintains the recurrent state through low-rank updates and periodic recompression under a fixed storage budget. Its key-memory read determines each write residual, and updated factors produce query outputs. Our matched-storage comparisons use Static, calibrated Oja, and a GDN adaptation of COD.

6 Conclusion and Future Work

Our results suggest that recurrent-state compression should distinguish capacity from geometry. Having enough rank to approximate the current state does not ensure that the same subspace remains useful over time. FactorState turns this insight into a training-free representation that tracks the evolving GDN state under a fixed storage budget. The default configuration saves approximately 61% of recurrent-state storage, with task scores close to Dense across the evaluated benchmarks. Our CUDA implementation also reduces execution time and peak GPU memory on the evaluated workloads. This study focuses on GDNs, whose update structure enables recurrent reads and writes directly in factor space. Future work could investigate whether capacity–geometry decoupling also arises in other recurrent architectures and how FactorState could be adapted to their state representations and update rules.

REFERENCES

M. Brand. Fast Low-Rank Modifications of the Thin Singular Value Decomposition. *Linear Algebra and Its Applications*, 415(1):20–30, May 2006. doi: 10.1016/j.laa.2005.07.021. URL https://www.merl.com/publications/TR2006-059.

Loïc Cabannes, Pierre-Emmanuel Mazaré, Gergely Szilvasy, Matthijs Douze, Maria Lomeli, Ilze Amanda Auzina, Justin Carpentier, Gabriel Synnaeve, and Hervé Jégou. Sparse delta memory: Scaling the state of linear RNNs through sparsity, 2026. URL https://arxiv.org/abs/2607.07386.

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

Wanyun Cui. A hippocampus for linear attention: An exact memory for what the recurrent state forgets, 2026. URL https://arxiv.org/abs/2607.02303.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset, 2021. URL https://arxiv.org/abs/2103.03874.

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

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Junqing Lin, Jingwei Sun, and Guangzhong Sun. DeltaLog: Deferred materialization of recurrent states for linear attention decoding, 2026. URL https://arxiv.org/abs/2608.15533.

Ze-Wei Liou and Tri Dao. ReplaySSM: Cache SSM inputs, not state. Dao AI Lab technical article and vLLM RFC, 2026. URL https://dao-lab.ai/blog/2026/replayssm/. Accessed 2026-08-25.

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, 2023. URL https://arxiv.org/abs/2305.01210.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. MathVista: Evaluating mathematical reasoning of foundation models in visual contexts. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 23439–23554, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/663bce02a0050c4a11f1eb8a7f1429d3-Paper-Conference.pdf.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. ChartQA: A benchmark for question answering about charts with visual and logical reasoning, 2022. URL https://arxiv.org/abs/2203.10244.

William Merrill, Yanhong Li, Tyler Romero, Anej Svete, Caia Costello, Pradeep Dasigi, Dirk Groeneveld, David Heineman, Bailey Kuehl, Nathan Lambert, Chuan Li, Kyle Lo, Saumya Malik, DJ Matusz, Benjamin Minixhofer, Jacob Morrison, Luca Soldaini, Finbarr Timbers, Pete Walsh, Noah A. Smith, Hannaneh Hajishirzi, and Ashish Sabharwal. OLMo hybrid: From theory to practice and back, 2026. URL https://arxiv.org/abs/2604.03444.

Youssef Mroueh, Etienne Marcheret, and Vaibhava Goel. Co-Occurring Directions Sketching for Approximate Matrix Multiply. In Aarti Singh and Jerry Zhu (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 567–575. PMLR, 2017. URL https://proceedings.mlr.press/v54/mroueh17a.html.

Philipp Nazari and T. Konstantin Rusch. The key to state reduction in linear attention: A rank-based perspective. In *New Frontiers in Associative Memory Workshop at ICLR*, 2026. doi: 10.48550/ARXIV.2602.04852. URL https://openreview.net/forum?id=GfjoMXfaXq.

Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273, 1982. doi: 10.1007/BF00275687.

Yuqi Pan, Yongqi An, Zheng Li, Yuhong Chou, Ruijie Zhu, Xiaohui Wang, Mingxuan Wang, Jinqiao Wang, and Guoqi Li. Scaling linear attention capacity with sparse state expansion. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=R6DrJ4tnGV.

Qwen Team. Qwen3.5-4B. Hugging Face model card, 2026a. URL https://huggingface.co/Qwen/Qwen3.5-4B. Accessed 2026-08-25.

Qwen Team. Qwen3.5-9B. Hugging Face model card, 2026b. URL https://huggingface.co/Qwen/Qwen3.5-9B. Accessed 2026-09-14.

Xingyu Shen, Yingfa Chen, Zhen Leng Thai, Xu Han, Zhiyuan Liu, and Maosong Sun. StateX: Enhancing RNN recall via post-training state expansion. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 21339–21353. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1073. URL https://aclanthology.org/2026.findings-acl.1073/.

Ao Sun, Hongtao Zhang, Heng Zhou, Yixuan Ma, Yiran Qin, Tongrui Su, Yan Liu, Zhanyu Ma, Jun Xu, Jiuchong Gao, Jinghua Hao, and Renqing He. State rank dynamics in linear attention LLMs, 2026. URL https://arxiv.org/abs/2602.02195.

Li Wang, Yi Su, Xiabao Wu, Chiran You, Yongchao Liu, Zhan Qiu, Juelu Zhang, Jiajun Zheng, Fangxin Liu, Jie Zhang, Chen Tian, and Chengying Huan. Bole: Efficient tree speculation for hybrid-attention language models, 2026. URL https://arxiv.org/abs/2608.01651.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/4904fad153f6434a7bcf04465d4be2cc-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.

Longwei Zou and Lin Zhong. KVBuffer: IO-aware serving for linear attention, 2026. URL https://arxiv.org/abs/2605.19049.

Artificial intelligence (AI) use statement

AI tools assisted with writing and debugging portions of the experimental code, searching for relevant literature, and preparing the manuscript through drafting, revision suggestions, and grammatical corrections. The authors directed the research and take full responsibility for the code, results, references, and final manuscript.

A Capacity–geometry diagnostics

A.1 Multi-rank evaluation protocol

The Qwen evaluation contains 16 streams, four each from natural language, retrieval, mathematics, and code, each with 512 dense-prefill tokens and 512 teacher-forced updates. Ranks $8, 16, 24, 32$ share an FP32 decomposition at anchors $64, 128, 256, 384$ and ages $0, 1, 2, 4, 8, 16, 32, 64, 128$. Query reads follow the update; key reads use the decayed state before the write. All GDN layers and value

heads are observed. Figure 1 and Table 3 pool energy sums across streams, layers, heads, and all four anchors at each age; the update-384 check restricts pooling to that anchor.

OLMo uses 16 streams, eight natural and eight retrieval, with the same ranks, anchors, ages, read branches, and statistics, covering all 24 GDN layers and 30 value heads. These streams are fixed before metric inspection and uniformly decomposed in FP32 with the `gesvd` driver. Observation hooks were checked against uninstrumented dense execution: logits and final recurrent states agree exactly.

A.2 Metrics and statistical procedures

Confidence intervals use 10,000 percentile-bootstrap resamples of whole streams. Metrics pool squared errors and reference energies before taking ratios or square roots.

Using the branch states and reads from Equation 3, we measure stale read error at anchor a, recurrent age $\tau = t - a$, and rank r as

$$E^{(b)}_{a,\tau,r} = \left(\frac{\sum_{i \in \mathcal{I}^{(b)}_{a,\tau}} \| y^{(b)}_i - y^{(b,\mathrm{fr})}_{a,i,r} \|_2^2}{\sum_{i \in \mathcal{I}^{(b)}_{a,\tau}} \| y^{(b)}_i \|_2^2} \right)^{1/2}. \tag{8}$$

Here $b \in \{q, k\}$ selects the query or key branch, and $\mathcal{I}^{(b)}_{a,\tau}$ contains nested layer/head observations. Resamples recompute pooled energies; individual observations are not independent cases.

Capacity change is small when its two-sided 95% confidence interval (CI) lies in $[-0.005, +0.005]$; capacity sufficiency additionally requires a current squared residual below 0.05. Geometry and read aging use one-sided 95% lower bounds; secondary-rank tests use Holm correction.

A.3 Multi-rank results and later-anchor checks

All evaluated Qwen ranks pass the capacity and Holm-corrected aging tests (Table 3). OLMo ranks 16, 24, and 32 do likewise; rank 8 is capacity-limited, with current squared residual 0.093938. At update-384 anchors, Qwen retains the same trends in capacity, geometry, and both read errors; the OLMo rank-16 check retains these directions.

Table 3: Multi-rank Qwen and OLMo geometry robustness, each on 16 streams. Contrasts are age 128 minus age 1. Below each estimate, whole-stream bootstrap uncertainty is a two-sided 95% interval for ΔC or a one-sided 95% lower bound (parentheses) for geometry and query/key reads.

Rank	Capacity loss C at age 128	ΔC [95% CI]	Δ geometry (lower bound)	Δ query read (lower bound)	Δ key read (lower bound)
Qwen3.5-4B: all evaluated ranks capacity-eligible					
8	0.026899	0.001366 [0.001149,0.001553]	0.046378 (0.043818)	0.046112 (0.042615)	0.091734 (0.083841)
16	0.010007	0.000606 [0.000523,0.000681]	0.039380 (0.036859)	0.042331 (0.039010)	0.084170 (0.076659)
24	0.004631	0.000325 [0.000286,0.000362]	0.033316 (0.031116)	0.038052 (0.034876)	0.075099 (0.068083)
32	0.002356	0.000185 [0.000165,0.000205]	0.028116 (0.026159)	0.034733 (0.031740)	0.067344 (0.060821)
OLMo Hybrid 7B: rank 8 capacity-limited/ineligible; ranks 16, 24, and 32 eligible					
8	0.093938	0.003855 [0.003021,0.004662]	0.069139 (0.060165)	0.155492 (0.132137)	0.121379 (0.101554)
16	0.041184	0.002237 [0.001684,0.002786]	0.068518 (0.058878)	0.144635 (0.121371)	0.122254 (0.101914)
24	0.021178	0.001352 [0.001016,0.001687]	0.063758 (0.054381)	0.138037 (0.116213)	0.117953 (0.098314)
32	0.011592	0.000817 [0.000620,0.001015]	0.058200 (0.049448)	0.130884 (0.109821)	0.110586 (0.091793)

B FACTORSTATE IMPLEMENTATION DETAILS

B.1 CUDA CHOLESKY-CORE RECOMPRESSION

Our CUDA implementation receives expanded factors U', V' with c columns. We write $\mathrm{sym}(A) = (A + A^\top)/2$ and use I_c for the $c \times c$ identity. For each head, the diagonal regularization terms λ_U, λ_V equal $8192\,\epsilon_{32}$ times the largest absolute diagonal entry of the corresponding symmetric Gram matrix, with unit scale when that maximum is zero. Here $\epsilon_{32} = 2^{-23}$ is the FP32 machine epsilon. The recompressed factors are computed as

$$\begin{aligned} H_U &= \mathrm{sym}(U'^\top U') + \lambda_U I_c = L_U L_U^\top, \\ H_V &= \mathrm{sym}(V'^\top V') + \lambda_V I_c = L_V L_V^\top, \\ K = L_U^\top L_V, &\qquad (P_K, \Sigma, Q_K) = \mathrm{JacobiSVD}(K), \\ X_U = L_U^{-\top}(P_K)_{:r}, &\qquad X_V = L_V^{-\top}(Q_K)_{:r}, \\ \widetilde{U} = (U' X_U)\Sigma_r^{1/2}, &\qquad \widetilde{V} = (V' X_V)\Sigma_r^{1/2}. \end{aligned} \tag{9}$$

Here L_U, L_V are lower-triangular Cholesky factors; $K \in \mathbb{R}^{c\times c}$ is the coordinate core; P_K, Q_K contain its left and right singular vectors; and Σ contains its singular values. The block $\Sigma_r \in \mathbb{R}^{r\times r}$ retains the leading singular values. The maps X_U, X_V return the leading r directions, denoted by $: r$, to factor coordinates. The committed factors $\widetilde{U}, \widetilde{V}$ share the square-root singular-value scaling. For r16/B8, $c = 25$ and $r = 16$. Our CUDA implementation performs maintenance in three stages: Gram/Cholesky/core construction, warp-level Jacobi decomposition and coordinate solves, and factor reconstruction with state commitment. Jacobi rotations use a pairwise threshold of 10^{-6}; convergence requires normalized off-diagonal Gram residual at most 10^{-5}, with at most 16 sweeps. Reusable PyTorch buffers provide the maintenance workspace; recompression uses no external solver workspace. Both the CUDA benchmark and performance results use these CUDA updates and recompression; Appendix E.4 describes their use in continuous batching.

For zero-indexed GDN layer ℓ, a phase offset $p_\ell = \ell \bmod (B + 1)$ staggers recompression. Initialization adds p_ℓ zero columns, setting the initial live-column count to $c_0 = r + p_\ell$ without changing the represented matrix. The first recompression follows $B + 1 - p_\ell$ writes, then recurs every $B + 1$ writes (Equation 5).

B.2 MODEL CONFIGURATIONS AND EXECUTION PATH

Qwen3.5-4B and OLMo-Hybrid-Instruct-SFT-7B each have 24 GDN layers. Qwen uses 128×128 states with 32 heads; OLMo uses 96×192 states with 30 value heads. Model-specific adapters leave BF16 weights, attention caches, and dense prefill unchanged. Each selected prefill state is converted once, after which the systems implementation releases its dense storage. Mechanism experiments use the FP32 PyTorch reference path; cross-model benchmarks use Triton. CUDA quality and systems comparisons use the same FLA convolution for both methods. Quality and mechanism experiments run on RTX 3090.

C STATE-REPRESENTATION EVALUATION

C.1 BASELINES AND MATCHED-MEMORY SETUP

The matched-memory Qwen corpus has 24 fixed 1,024-token streams, with 12 calibration and 12 held-out validation cases balanced across natural/retrieval/math/code. For the rank-23 Oja control, calibration considers learning rates $\{.001, .003, .01, .03, .1\}$ and periods $\{1, 8\}$. Minimizing pooled query-read NRMSE on the calibration split selects $\eta = 0.001$ and $p = 8$; these settings are fixed for GPU validation. Every conditional method consumes the same stored dense (q, k, v, g, β) trace in 32-bit floating point (FP32).

Oja control. We adapt Oja's rule (Oja, 1982) to $\widehat{S}_{t-1} = LZR^\top$, with orthonormal rank-23 bases L, R and a 23×23 core Z. Static and Oja use rank-23 SVD initialization of the dense prefill state.

Before writing, the represented state gives

$$\delta_t^{\mathrm{O}} = \beta_t\big(v_t - \alpha_t R Z^\top L^\top k_t\big)\,.$$

Every $p = 8$ writes, starting at write 8, the left basis observes the normalized key k_t; the right observes δ_t^{O} without further normalization. With $\eta = 0.001$,

$$\begin{aligned}
\mathcal{U}_\eta(Q, x) &= \mathrm{qf}\big(Q + \eta(I - QQ^\top)xx^\top Q\big)\,,\\
L_t = \mathcal{U}_\eta(L, k_t), &\qquad R_t = \mathcal{U}_\eta(R, \delta_t^{\mathrm{O}}),\\
Z_t &= \alpha_t(L_t^\top L)Z(R^\top R_t) + (L_t^\top k_t)(R_t^\top \delta_t^{\mathrm{O}})^\top.
\end{aligned}$$

Here qf is thin QR with signs chosen to make the triangular diagonal positive. The core projects the decayed previous state and the complete current write into the updated bases. On other steps, $L_t = L$, $R_t = R$, and $Z_t = \alpha_t Z + (L^\top k_t)(R^\top \delta_t^{\mathrm{O}})^\top$. The query reads $R_t Z_t^\top L_t^\top (q_t/\sqrt{d_k})$ after this update. Static uses this fixed-basis recurrence throughout. Both controls persist only their bases and core.

COD control. We adapt Co-Occurring Directions (Mroueh et al., 2017) to the GDN recurrence, representing $\widehat{S} = XY^\top$ with 25 columns per factor. Rank-23 SVD initializes balanced factors, leaving two zero columns; FactorState uses rank-16 initialization. COD computes $\delta_t^{\mathrm{C}} = \beta_t(v_t - \alpha_t \widehat{S}_{t-1}^\top k_t)$, multiplies both factors by $\sqrt{\alpha_t}$, and inserts $(k_t, \delta_t^{\mathrm{C}})$ into one paired vacant column. Like FactorState, it reads the query after writing and before maintenance. When all columns are occupied, it computes

$$\begin{aligned}
X = Q_X R_X, \quad Y = Q_Y R_Y, \quad R_X R_Y^\top = U\Sigma V^\top,\\
\widetilde{\Sigma}_{ii} = \max(\Sigma_{ii} - \sigma_{13}, 0), \qquad X \leftarrow Q_X U\widetilde{\Sigma}^{1/2}, \quad Y \leftarrow Q_Y V\widetilde{\Sigma}^{1/2}.
\end{aligned}$$

The 13th singular value fixes the median rule for our odd capacity of 25; the original algorithm uses an even capacity. Vacant columns are managed separately for each matrix. This is a feedback-recurrence adaptation, not a direct application of the original matrix-product approximation guarantee. FactorState and COD use column-scaled QR on GPU: scale each column by its maximum magnitude before QR and restore the scale in R. The represented matrices and truncation rules are unchanged, and arithmetic remains FP32.

For equal storage dtype, Static/Oja use $(d_k r + d_v r + r^2)/(d_k d_v)$ of Dense state bytes, whereas COD and FactorState count all physical factor columns. Diagnostic counters and QR/SVD workspace are separate from representation bytes; COD's compression counter uses one 64-bit integer per matrix. This experiment makes no runtime or peak-memory comparison.

Table 4: Physical state representations relative to Dense FP32. FactorState counts buffered columns and its current write slot. COD counts all 25 paired columns. Diagnostic counters and temporary QR/SVD workspace are excluded from representation bytes.

Method	State bytes (%)	Representation
Dense GDN	100.0000	Dense recurrent matrix
FactorState r16/B8	39.0625	25 physical columns
Persistent r24/B0	39.0625	25 physical columns
Static r23	39.1663	Fixed bases and a 23×23 core
Oja r23	39.1663	Updated bases and a 23×23 core
COD25	39.0625	Paired sketch; rank-23 initialization

C.2 Error metrics and paired comparisons

State error uses the terminal state after maintenance at update 512. Query references are $y_t^{(q)} = S_t^\top(q_t/\sqrt{d_k})$, and key references are $y_t^{(k)} = \alpha_t S_{t-1}^\top k_t$. Both read metrics cover all 512 updates and all 24 layers and 32 heads. Approximate key reads precede each write and are not query-scaled. Native query reads follow the method definitions above; query reads after maintenance are also recorded separately and are not used for Figure 3.

For method M and observations $i \in \mathcal{J}$, the state and read errors are

$$E_{\text{state}}(M) = \left(\frac{\sum_{i \in \mathcal{J}} \|S_i - \widehat{S}_i^{(M)}\|_F^2}{\sum_{i \in \mathcal{J}} \|S_i\|_F^2} \right)^{1/2},$$
$$E_{\text{read},b}(M) = \left(\frac{\sum_{i \in \mathcal{J}} \|y_i^{(b)} - \widehat{y}_i^{(M,b)}\|_2^2}{\sum_{i \in \mathcal{J}} \|y_i^{(b)}\|_2^2} \right)^{1/2}, \quad b \in \{q, k\}. \tag{10}$$

We draw 10,000 paired bootstrap resamples of the 12 complete validation streams, sharing indices across methods and metrics. Figure error bars are the 2.5th and 97.5th percentiles of the recomputed pooled errors; layers, heads, and updates are not independent replicates. For control C, we report the pooled error ratio and the geometric mean of per-stream ratios. For each stream j, define the log ratio

$$\ell_j(M, C) = \log\left(\frac{E_j(M)}{E_j(C)} \right). \tag{11}$$

The geometric ratio is obtained by exponentiating the mean ℓ_j. Its interval exponentiates the percentiles of the resampled mean, and differs from the pooled-ratio intervals and absolute error bars. COD has the lowest pooled error among the three controls for every metric. FactorState has lower errors on all 12 streams; Tables 5 and 6 give the values and uncertainty.

Table 5: Matched-memory conditional fidelity on 12 validation streams. Values are pooled errors; brackets give two-sided 95% whole-stream bootstrap intervals. All methods use FP32.

Method	Final state	Query NRMSE	Key NRMSE
Static r23	0.325427 [0.305770, 0.344732]	0.118115 [0.109368, 0.126655]	0.159445 [0.153138, 0.165129]
Oja r23	0.325014 [0.305285, 0.344313]	0.118245 [0.109480, 0.126781]	0.159359 [0.153055, 0.165045]
COD 25 columns	0.275422 [0.271007, 0.280026]	0.038427 [0.036341, 0.040542]	0.054434 [0.052689, 0.056209]
FactorState r16/B8	0.110312 [0.108002, 0.112376]	0.011674 [0.010844, 0.012435]	0.017748 [0.016848, 0.018607]

Table 6: FactorState/COD error ratios with paired 95% whole-stream bootstrap intervals. The pooled ratio compares aggregate errors; the geometric ratio exponentiates the mean per-stream log ratio.

Metric	Pooled ratio	Geometric ratio
Final state	0.4005 [0.3950, 0.4060]	0.4005 [0.3950, 0.4059]
Query read	0.3038 [0.2939, 0.3124]	0.3022 [0.2916, 0.3120]
Key read	0.3260 [0.3173, 0.3337]	0.3250 [0.3163, 0.3333]

C.3 Long-horizon conditional replay

The 4,096-update conditional experiment uses eight Qwen3.5-4B streams, two per category, with 512 dense-prefill tokens and 4,096 teacher-forced updates. Query-read error is recorded every 32 steps and state error every 128 steps. Dense and factorized states receive the same realized GDN inputs.

State error rises from 0.119178 at update 1,024 to 0.136437 at update 4,096; query-read NRMSE rises from 0.012722 to 0.013953. The geometric means of per-stream 4,096-to-1,024 error ratios are 1.144 (95% CI $[1.096, 1.190]$) and 1.100 ($[1.042, 1.156]$), respectively. Intervals use 10,000 whole-stream bootstrap resamples.

D BENCHMARK EVALUATION

D.1 INPUTS, GENERATION, AND SCORING

All full-model comparisons pair Dense and FactorState on the same prompt, case-specific generation cap, and scorer. Unless stated below, generation uses BF16 weights, the non-thinking chat template, greedy argmax decoding, and natural end-of-sequence termination. We report each task's native score and do not pool scores across tasks. Exact seeds for data selection, generation, bootstrap resampling, and system workloads are recorded in the accompanying configuration file `seed_settings.json`.

The cross-model results in Table 1 evaluate r16/B8 using the Triton implementation described in Appendix B.2, separately from the Qwen3.5-4B CUDA benchmark.

Text benchmarks. In the CUDA benchmark, Dense and FactorState generate outputs independently using SDPA, FLA convolution, and KV reserved for each input and its output cap. The reported benchmarks contain 541 IFEval prompts (Zhou et al., 2023) (cap 1,280); 300 RULER cases (Hsieh et al., 2024) from common-word extraction, frequent-word extraction, question answering, multi-query needle retrieval, and multi-value needle retrieval at 4K/16K (30 cases per task–length cell); 300 GSM8K problems (cap 2,048); 500 MATH-500 problems (cap 8,192); and 542 EvalPlus problems (cap 768). Output caps include the first token produced by prefill. These 2,183 text instances, together with the 3,500 image questions described below, give 5,683 paired benchmark instances.

For the Triton evaluations, Qwen3.5-4B uses 51 RULER task–length cells with 50 cases each: 12 tasks at 4K/8K/16K/32K, plus `qa_2` at 8K/16K/32K. The `qa_2` 4K generator cannot fit its ten required documents and is excluded without replacement. The Qwen3.5-9B RULER comparison uses a fixed 500-case hard subset. RULER uses normalized substring matching averaged over required answers, with task-specific caps: 128 tokens for needle retrieval, 120 for common-word extraction, 50 for frequent-word extraction, 30 for variable tracking, and 32 for question answering. The OLMo subset has ten tasks at 4K/8K/16K, six cases per cell (180 total; seed 20260816).

The text mathematics suite interleaves 300 GSM8K (Cobbe et al., 2021) test problems sampled without replacement and all 500 MATH-500 test problems, the held-out MATH subset (Hendrycks et al., 2021; Lightman et al., 2023) released with PRM800K. We construct the suite with base seed 20260817 and sample GSM8K with seed 20260818. All problems use a common step-by-step prompt. The cross-model and diagnostic mathematics evaluations use a 2,048-token cap for both datasets. Correctness is computed with `math_verify`; parsing failures count as incorrect and are recorded. The upstream MATH-500 revision identifier is unavailable.

EvalPlus (Liu et al., 2023) contains all 164 HumanEval+ v0.1.10 and 378 Mostly Basic Python Problems Plus (MBPP+) v0.2.0 tasks, with one greedy sample per task and official Plus pass@1 scoring in an isolated evaluator. IFEval uses the official scorer, with strict prompt accuracy as the reported metric.

Multimodal benchmarks. The multimodal evaluations use the complete ChartQA (Masry et al., 2022) test split (2,500 questions, 1,250 human-written and 1,250 machine-generated) and MathVista (Lu et al., 2024) testmini (1,000 questions, 540 multiple-choice and 460 open-ended). Each has Qwen3.5-4B/9B Triton comparisons with PyTorch convolution and dynamic KV, plus an independent Qwen3.5-4B CUDA comparison with FLA convolution and preallocated KV. All use batch size one, SDPA, and the local processors' default image scaling, without an additional pixel limit. Images and processed input tokens and grids are held fixed within each comparison.

ChartQA appends a short-answer-only instruction to the question and uses a 100-token output cap. Relaxed accuracy follows the Pix2Struct reference: numerical predictions allow 5% relative error using the reference as denominator, preserving percentage conversion and zero-reference string matching; other answers are matched without case sensitivity. Only leading and trailing whitespace is stripped from the complete generated response.

MathVista retains the official query and requests brief necessary steps followed by `FINAL ANSWER: <answer>` on the last line, with a 2,048-token cap. We use *rule-based answer extraction with official normalization and scoring*, not model-assisted answer extraction. The final-line marker is case-insensitive; without it, the whole response must be a valid short answer. Choices accept a valid

letter, parenthesized letter, or exact option text; open answers accept finite numbers or safely parsed numeric lists. We then apply official option mapping, integer conversion, precision-based rounding, list comparison, and exact answer matching. Empty or invalid extractions count as incorrect; no number is guessed from the reasoning text and no ChartQA numerical tolerance is applied.

ChartQA has no truncated responses. MathVista truncation rates are 5.7%/5.5% for 4B Triton, 6.8%/6.9% for 9B Triton, and 4.7%/6.2% for 4B CUDA (Dense/FactorState). Corresponding extraction-failure rates are 6.5%/5.9%, 7.1%/7.5%, and 5.4%/7.0%. Truncation means reaching the output cap without EOS; these outputs remain in the scores.

D.2 Paired uncertainty estimates

Unless stated otherwise, intervals use 10,000 percentile-bootstrap resamples of paired cases, keeping Dense and FactorState scores for each case together. All seven CUDA benchmarks report two-sided 95% intervals (Table 8); IFEval, RULER, and EvalPlus also report one-sided 95% lower bounds for their prespecified margins. ChartQA resamples within human/machine groups and MathVista within its five task categories, preserving their proportions and keeping Dense/FactorState answers paired. Cross-model text mathematics resamples within fixed source and Dense-length strata, independent of FactorState outputs. Descriptive secondary intervals are not multiplicity-adjusted. The benchmark-response diagnostic is specified in Appendix D.5.

Prespecified score margins are -2 percentage points for IFEval and EvalPlus, and -1 point for RULER; their paired lower bounds are reported in Appendix D.4. Close scores or confidence intervals crossing zero do not establish equivalence; non-inferiority is a separate, margin-specific claim. Positive score differences are treated as trajectory variation rather than evidence of improved capability.

D.3 Triton benchmark results

Table 7 gives paired confidence intervals for the Triton results in Table 1.

Table 7: Cross-model paired score differences and two-sided 95% intervals (pp). Absolute scores and paired-case counts are in Table 1.

Model	Benchmark	Δ (pp)	95% CI
Qwen3.5-4B	IFEval strict prompt	-0.370	$[-2.588, +1.848]$
	RULER (51 cells)	-0.001	$[-0.111, +0.120]$
	GSM8K/MATH-500	-0.125	$[-1.875, +1.625]$
	EvalPlus	$+0.923$	$[-1.107, +2.952]$
	ChartQA test	-0.080	$[-0.200, +0.000]$
	MathVista testmini	$+0.800$	$[-0.600, +2.200]$
Qwen3.5-9B	RULER hard subset	$+0.080$	$[-0.497, +0.783]$
	GSM8K/MATH-500	-1.375	$[-2.875, +0.125]$
	EvalPlus	$+0.369$	$[-1.476, +2.214]$
	ChartQA test	-0.120	$[-0.400, +0.120]$
	MathVista testmini	$+0.900$	$[-0.600, +2.400]$
OLMo Hybrid 7B	RULER subset	-0.185	$[-1.463, +0.926]$
	GSM8K/MATH-500	-0.750	$[-2.750, +1.250]$
	EvalPlus	-0.738	$[-2.214, +0.738]$

D.4 CUDA benchmark results

Table 8 reports scores and two-sided intervals for all seven benchmarks. The one-sided 95% lower bounds are -1.294, -0.550, and -2.030 pp for IFEval, RULER, and EvalPlus, respectively. These support the prespecified -2 and -1 pp margins for IFEval and RULER, but not the -2 pp margin for EvalPlus. Mathematics and multimodal endpoints have no prespecified non-inferiority margins.

At a 2,048-token output cap, MATH-500 accuracy is 82.6% for Dense and 80.0% for FactorState, a difference of -2.6 percentage points (95% CI $[-5.6, +0.2]$). This comparison uses the same 500

problems as the 8,192-token result and is not counted as additional independent instances. Its intervals follow the paired bootstrap protocol in Appendix D.2.

Table 8: Qwen3.5-4B CUDA benchmark quality on 5,683 paired instances. Scores are percentages; Δ is FactorState minus Dense (pp), with two-sided 95% paired intervals. GSM8K/MATH-500 output caps are 2,048/8,192 tokens. ChartQA and MathVista use image–text inputs.

Benchmark	n	Dense	FactorState	Δ (pp)	95% CI
IFEval strict prompt	541	82.255	82.994	$+0.739$	$[-1.664, +3.327]$
RULER 4K/16K	300	87.400	87.117	-0.283	$[-0.600, +0.000]$
GSM8K	300	93.000	93.000	$+0.000$	$[-1.667, +2.000]$
MATH-500	500	93.800	92.000	-1.800	$[-4.200, +0.400]$
EvalPlus	542	71.587	71.402	-0.185	$[-2.214, +2.030]$
ChartQA test	2,500	84.480	84.480	$+0.000$	$[-0.160, +0.160]$
MathVista testmini	1,000	78.600	77.900	-0.700	$[-2.200, +0.800]$

D.5 Output-distribution diagnostics

We sample 400 of the CUDA text evaluation's 2,183 questions: 100 each from instruction following, retrieval, mathematics, and code (seed 20260912), without using scores or KL. Stratification gives ten RULER cases per task–length cell, 38 GSM8K/62 MATH-500 problems, and 30 HumanEval+/70 MBPP+ problems. The CUDA runtime, generation, and scoring settings follow Appendix D.1; both mathematics datasets use a 2,048-token cap for this diagnostic. The rank/buffer study in Appendix E.3 uses these same 400 questions; they are not an independent validation set for that study.

After dense prefill, Dense and FactorState replay the saved greedy response under identical Dense-generated prefixes, maintaining independent states. A length-L response gives $L-1$ comparisons, excluding the prefill output and including terminal EOS prediction when present. When replayed on its own generated prefixes, each method's greedy next-token predictions match the saved tokens.

Full-vocabulary $\mathrm{KL}(p^{\mathrm{Dense}} \| p^{\mathrm{FactorState}})$ is measured in nats (FP32, temperature 1). Task scores use independent generations and are reported in Table 14; forced replay is not scored as a new task response. As a complementary check, we also evaluate under FactorState-generated prefixes; the corresponding mean KL values are 0.00974, 0.00107, 0.00572, and 0.00352 nats for IFEval, RULER, mathematics, and EvalPlus, respectively.

Table 9: Qwen3.5-4B output-distribution diagnostics under Dense-generated prefixes. Metrics are averaged within each question and then across questions. Task scores for the same cases appear in Table 14.

Benchmark	n	Mean KL (nat)	Top-1 agreement (%)
IFEval	100	0.00974	96.09
RULER	100	0.00105	99.63
GSM8K/MATH-500	100	0.00619	98.58
EvalPlus	100	0.00347	99.00

E Systems evaluation

E.1 Fixed-active-batch runtime

A paired run comprises separate Dense and FactorState processes using Qwen3.5-4B BF16, SDPA, FLA convolution, preallocated KV, and unique 64-token prompts. FactorState uses CUDA FP32 r16/B8 with the implementation described in Appendix B.1. Prefill microbatches contain 32 sequences; all sequences remain active, ignoring EOS. Execution order is counterbalanced. Independent warmup runs for 18 steps.

The evaluation uses 12 paired runs (24 processes): four each for RTX 3090 concurrency 192 and A800 concurrency 512, and two each for A800 concurrency 192 and 384. Statistics use independent

Table 10: Timing decomposition, Dense → FactorState. Admission and decode are synchronized wall seconds; P50/P99 are complete-step GPU-event milliseconds. Entries are medians across processes.

GPU / C	Admission (s)	Decode (s)	P50 (ms)	P99 (ms)
3090 / 192	$1.81 \to 3.82$	$46.72 \to 33.65$	$91.25 \to 65.86$	$93.34 \to 69.14$
A800 / 192	$0.67 \to 2.02$	$97.64 \to 86.64$	$47.74 \to 42.17$	$51.92 \to 44.65$
A800 / 384	$1.31 \to 4.02$	$165.19 \to 123.40$	$80.55 \to 60.07$	$89.44 \to 70.68$
A800 / 512	$1.73 \to 5.35$	$222.12 \to 164.02$	$108.29 \to 80.01$	$119.53 \to 93.63$

paired runs under these conditions, not individual decode steps. RTX 3090 uses 512 decode steps and A800 uses 2,048, with KV capacity reserved for the respective request lengths.

Complete-request timing includes prefill, KV reservation, factor conversion, cache assembly, decode, token selection, and maintenance; model loading, tokenization, and independent warmup are excluded. Admission ends when the cache is ready; decode throughput uses synchronized wall time thereafter. Step P50/P99 use GPU events including maintenance. Component medians need not sum to the request median. Tables 10–11 report timing and short-request behavior.

Table 11: Short-request boundary on A800 at concurrency 192. Entries are cumulative complete-request time ratios (FactorState/Dense), including admission, for two paired runs. Each checkpoint uses KV reserved for the full 2,048-step horizon.

GPU / C	Pair	270	512	768	1,024	2,048
A800 / 192	1	1.052026	0.991751	0.962013	0.942217	0.894600
A800 / 192	2	1.070250	1.008591	0.978435	0.958801	0.909043

The request-ratio ranges are $[0.771585, 0.772846]$, $[0.894600, 0.909043]$, $[0.764528, 0.766098]$, and $[0.756599, 0.757146]$ in main-table order. Gains are computed from median paired ratios, not from the ratio of per-method median times. A800 concurrency 192 is slower at 270 steps in both paired runs, mixed at 512, and faster in both at measured checkpoints from 768 onward. Checkpoints use the KV reservation for the full horizon, not separately reserved shorter requests. These checkpoints do not establish an exact crossover.

E.2 Memory decomposition

Peak allocated memory is in Table 2; Table 12 adds reserved memory. Both peaks include admission, decode, conversion, and workspace.

All 25 columns use 39.0625% of same-dtype Dense recurrent-state bytes, saving 60.9375%. At A800 concurrency 512, recurrent storage falls from 24 to 9.375 GiB; both methods retain 33 GiB of KV. Weights, convolution state, KV, and other allocations reduce total peak savings. Peak allocation includes FactorState's 167.312 MiB maintenance scratch as model workspace. The implementation uses no external solver workspace.

Table 12: Peak PyTorch reserved memory (GiB) over the complete request, including reusable allocator blocks. Peak allocated memory is reported in Table 2.

GPU / C	Dense	FactorState
3090 / 192	23.08	18.45
A800 / 192	32.88	27.40
A800 / 384	55.68	45.00
A800 / 512	70.28	55.63

E.3 RANK AND BUFFER CONFIGURATIONS

This experiment compares Dense, r16/B0, r16/B8, and r24/B0 on Qwen3.5-4B with BF16 weights, SDPA, FLA convolution, and preallocated KV. All factor configurations use the CUDA FP32 implementation and solver settings in Appendix B.1. Including the write slot, r16/B0 reserves 17 columns (26.5625% of same-dtype Dense); r16/B8 and r24/B0 reserve 25 (39.0625%). With layer-staggered maintenance, B0 recompresses every update and B8 every nine.

Performance uses RTX 3090 at concurrency 192, with 64-token prompts, 512 decode steps, KV capacity 576, and all sequences active. Two counterbalanced blocks each run four independent processes, ordered Dense/r16B0/r16B8/r24B0 and then reversed. Comparisons pair configurations within a block. Timing and independent 18-step warmup follow Appendix E.1. Table 13 summarizes complete measured runs separately from Table 2. The paired r16/B8 request-time ratios are 0.7631 and 0.7618.

Table 13: CUDA rank/buffer comparison on RTX 3090 ($C = 192$, $T = 512$). Values are two-block means; Δ time averages paired percentage changes from Dense. Allocated/reserved memory is peak GiB; State is percent of same-dtype Dense.

Config.	Request (s)	Δ time (%)	Decode tokens/s	Allocated	Reserved	State (%)
Dense	47.56	+0.00	2147	21.61	23.14	100.00
r16/B0	50.67	+6.52	2094	14.74	17.55	26.56
r16/B8	36.27	-23.75	3022	15.92	18.52	39.06
r24/B0	73.93	+55.43	1401	15.93	18.50	39.06

Quality uses the same frozen 400 questions as Appendix D.5, with 100 per task; these add no instances to the 5,683 total. Four configurations yield 1,600 responses, sharing input tokens, per-case seeds, and output caps: 1,280 for IFEval, 32/50/120/128 for RULER, 2,048 for math, and 768 for code, including the prefill token. All use batch size one, greedy generation, disabled thinking, and natural EOS. Scoring follows Appendix D.1. Table 14 reports native task metrics without averaging across tasks.

Table 14: CUDA rank/buffer quality on the frozen 400-case subset, with 100 cases per task. Scores are percentages using IFEval strict prompt accuracy, native RULER scoring, math answer accuracy, and EvalPlus pass@1 with the extended tests.

Configuration	IFEval	RULER	GSM8K/MATH-500	EvalPlus
Dense	77.000	87.600	91.000	67.000
r16/B0	76.000	87.700	88.000	68.000
r16/B8	80.000	87.800	88.000	68.000
r24/B0	71.000	87.700	86.000	69.000

Table 15 uses 10,000 paired case-bootstrap resamples. IFEval resamples cases directly, RULER resamples within task–length strata, and math and code resample within their source datasets. Relative to r16/B0, r16/B8 scores 4 percentage points higher on IFEval and matches its math and code scores. Relative to Dense, however, r16/B8 scores 3 points lower on math, with a 95% CI of $[-8, 2]$ points. At the same state-storage budget, r24/B0 has lower IFEval and math scores than r16/B8, but a higher code score. These results show task-dependent rank–buffer trade-offs across the tested configurations.

Table 15: Paired quality differences on the same 400 cases. Each cell gives Δ (pp) and its two-sided 95% bootstrap interval, with contrasts taken in the order shown. Intervals are descriptive.

Contrast	IFEval	RULER	GSM8K/ MATH-500	EvalPlus
r16/B0 − Dense	-1.000 $[-8.000, +6.000]$	$+0.100$ $[-0.400, +0.700]$	-3.000 $[-8.000, +1.000]$	$+1.000$ $[-5.000, +7.000]$
r16/B8 − Dense	$+3.000$ $[-1.000, +8.000]$	$+0.200$ $[-0.200, +0.700]$	-3.000 $[-8.000, +2.000]$	$+1.000$ $[-5.000, +7.000]$
r24/B0 − Dense	-6.000 $[-12.000, -1.000]$	$+0.100$ $[-0.200, +0.500]$	-5.000 $[-10.000, +0.000]$	$+2.000$ $[+0.000, +5.000]$
r16/B8 − r16/B0	$+4.000$ $[+0.000, +9.000]$	$+0.100$ $[+0.000, +0.300]$	$+0.000$ $[-3.000, +3.000]$	$+0.000$ $[-5.000, +5.000]$
r16/B8 − r24/B0	$+9.000$ $[+4.000, +15.000]$	$+0.100$ $[+0.000, +0.300]$	$+2.000$ $[+0.000, +5.000]$	-1.000 $[-7.000, +5.000]$

E.4 Continuous batching

RTX 3090 processes 1,024 distinct 64-token prompts with at most 96 outstanding; A800 processes 4,096 with at most 512. Both methods use a synchronous FIFO scheduler with shared request order and budgets, replacing each request after completion and release. Output lengths 256/512/1,024/2,048 each occur 256 times on RTX 3090 and 1,024 on A800. Greedy generation ignores early EOS, producing 983,040/3,932,160 tokens including prefill outputs.

Both methods use Qwen3.5-4B BF16, FLA convolution, FlashInfer, 16-token KV pages, 2,112-token capacity per request, and prefill microbatches of up to 32, interleaved with decode. We capture GDN CUDA graphs on the eighth occurrence of each supported active-batch size: 1–16 plus 32/64/96 (RTX 3090) or 32/64/128/256/512 (A800), without padding. FactorState uses the FP32 r16/B8 implementation in Appendix B.1. Cross-layer recompression indices are stored in reusable pinned host-memory buffers between decode steps. Initialization uses layerwise FP32 Gram coordinates and GPU SVD fallback.

Synchronized timing spans memory-pool allocation through final release, including prefill, initialization and fallback, cache setup, decode, token selection, maintenance, and first graph capture. Model loading, input preparation, compilation, writing, and independent warmup are excluded. Warmup uses 32 requests and 18 steps, discarding its states. Decode-wall throughput divides post-prefill tokens (982,016/3,928,064) by decode wall time including maintenance. Memory peaks include graph pools, staging, factor conversion, and GPU workspace; host buffers are separate. Dense/FactorState states occupy 4.5/1.7578125 GiB on RTX 3090 and 24/9.375 GiB on A800; shared KV reservations are 6.1875 and 33 GiB, respectively.

Table 16 summarizes four independent paired runs per GPU with alternating method order. Completion-time ratios are 0.863307, 0.861159, 0.862212, 0.860510 for RTX 3090 and 0.844760, 0.840035, 0.838136, 0.853634 for A800.

Table 16: Per-method medians and median paired relative changes for continuous batching.

Metric	Dense	FactorState	Change
RTX 3090: at most 96 outstanding, 1,024 requests			
Workload completion time (s)	581.60	500.89	-13.83%
Decode-wall throughput (token/s)	1751.34	2133.51	$+21.83\%$
Peak allocated memory (GiB)	21.32	19.36	-9.21%
Peak reserved memory (GiB)	21.99	20.48	-6.88%
A800: at most 512 outstanding, 4,096 requests			
Workload completion time (s)	941.61	792.58	-15.76%
Decode-wall throughput (token/s)	4394.77	5697.25	$+29.57\%$
Peak allocated memory (GiB)	68.57	54.80	-20.08%
Peak reserved memory (GiB)	77.83	55.96	-28.10%