DeltaLog

Deferred Materialization of Recurrent States for Linear Attention Decoding

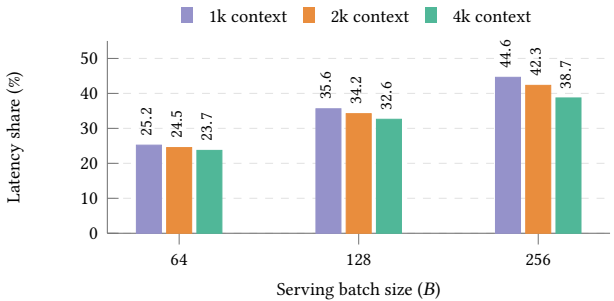
Paper authors: Junqing Lin, Jingwei Sun, Guangzhong Sun
University of Science and Technology of China

Research discussion · 10 October 2026
arXiv:2608.15533v1 · 22 main slides + 3 backup slides

Keep each logical state current; commit the dense state periodically.

https://arxiv.org/abs/2608.15533v1

A fixed-size state can still dominate decoding



The state-update tax

Compact factors: $O(d_k + d_v)$

Dense commit: $O(d_k d_v)$

At 1k context, the materialization share grows from 25.2% to 44.6% as batch size grows from 64 to 256.

Paper, Fig. 2, PDF p. 3. Qwen3.6-35B-A3B.

$B = 512$, $H = 32$, $d_k = d_v = 128$, FP32: **1 GiB of state per layer**; about 2 GiB for one dense read and write.

Paper, Secs. 2.3–2.4, PDF pp. 3–4.

The update is compact; the carried state is dense

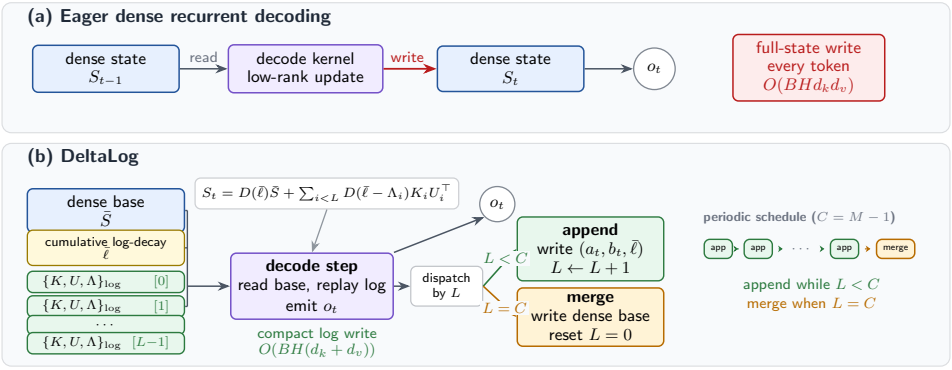
$$D(\ell) = \mathrm{diag}(e^{\ell}), \qquad S_t = D(\lambda_t) S_{t-1} + a_t b_t^{\top}$$

Quantity	Role
$S_t \in \mathbb{R}^{d_k \times d_v}$	Prefix memory for one head
λ_t	Scalar-per-head or key-wise log decay
$a_t \in \mathbb{R}^{d_k}$, $b_t \in \mathbb{R}^{d_v}$	Compact update factors

- Delta-rule updates construct b_t using an old-state readout.
- Output and correction reads must retain their operator-defined timing.

Paper, Eq. (3), Table 1, PDF p. 2; Algorithm 1, PDF p. 5.

Store a dense base plus a bounded update log



$M - 1$ append steps $\longrightarrow$ one merge step. Capacity $C = M - 1$.

Read the dense base every step; avoid writing it on append steps.

Paper, Fig. 4, PDF p. 5; Sec. 3, PDF p. 4. Original author figure.

Each logged update receives only later decay

$$S_t = D(\bar{\ell})\bar{S} + \sum_{i=0}^{L-1} D\left(\bar{\ell} - \Lambda_{\log}[i]\right) K_{\log}[i] U_{\log}[i]^\top$$

- $\bar{S}$: dense base at the last merge.
- $\bar{\ell}$: cumulative log decay since that merge.
- $\Lambda_{\log}[i]$: cumulative log decay when entry i was inserted.
- $K_{\log}[i], U_{\log}[i]$: the stored compact update factors.

Why the difference of snapshots?

$\bar{\ell} - \Lambda_{\log}[i]$ selects the decay accumulated *after* insertion. A new entry has zero later decay and therefore unit weight.

Paper, state invariant, Eq. (7), Sec. 3.2, PDF p. 6.

Append and merge preserve the same recurrence

First compute required old-state readouts; then set $\bar{\ell} \leftarrow \bar{\ell} + \lambda_t$ and form (a_t, b_t).

Append: $L < C$

Store $(a_t, b_t, \bar{\ell})$ at slot L.

Increment L; retain the base.

$$D(\lambda_t)D(\ell) = D(\ell + \lambda_t)$$

Old contributions decay correctly; the new one enters at weight 1.

Merge: $L = C$

Materialize old entries *and* the current update into $\bar{S}$.

Reset $\bar{\ell} = 0$ and $L = 0$.

$$S_t = \bar{S} \quad \text{after reset.}$$

The current update needs no extra log slot.

Exact-arithmetic equivalence follows by induction over both paths.

Paper, Algorithm 1 and Eqs. (4)–(7), PDF pp. 5–6.

Readouts can consume the log without a dense commit

For a state query $x \in \mathbb{R}^{d_k}$, expand the invariant:

$$x^\top S_t = (x \odot e^{\bar{\ell}})^\top \bar{S} + \sum_{i=0}^{L-1} c_i(x)\, U_{\log}[i]^\top,$$
$$c_i(x) = x^\top D\big(\bar{\ell} - \Lambda_{\log}[i]\big)\, K_{\log}[i].$$

- The base supplies long-range history; log entries supply recent updates.
- Each $c_i(x)$ is a scalar shared across all value coordinates.
- Query projections and Delta-rule correction projections can reuse this structure.

Teaching derivation from Eq. (7), PDF p. 6; replay projections in Sec. 3.3, PDF pp. 6–7.

Write savings compete with growing replay traffic

$$T_{\text{dense}} \approx T_{\text{common}} + 2C_{\text{state}},$$

$$T_{\text{delta}} \approx T_{\text{common}} + \underbrace{C_{\text{state}}}_{\text{base read}} + \underbrace{C_{\text{state}}/M}_{\text{base write}} + C_{\log}(M) + 2C_{\text{decay}},$$

$$C_{\log}(M) \approx \left(\frac{M-1}{M} + \frac{M-1}{2}\right) C_{\text{entry}}.$$

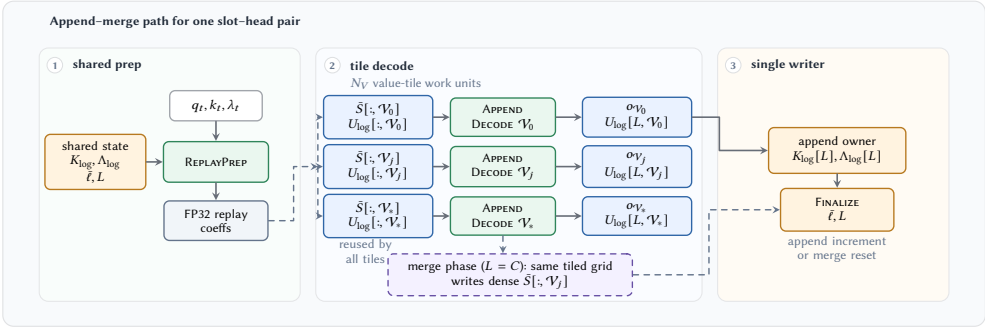
- Larger M: fewer merges, more live entries to replay.
- C_{entry}: bytes per log entry; C_{decay}: persistent decay bytes.

A derived state-traffic ratio

Ignoring log overhead, the state-traffic ratio is $2/(1 + 1/M)$: only $1.78\times$ at $M = 8$, even though dense writes fall $8\times$.

Paper, Eqs. (8)–(11), PDF p. 6. Single-read replay model; scratch and indexing omitted.

Tile values; share replay coefficients and metadata



- Each tile owns $\bar{S}[:, \mathcal{V}_j]$, its value log, and its output.
- One logical writer commits shared key factors, snapshots, and length.
- Merge uses the same tiled grid, with dense-base tiles as the target.

Paper, Fig. 5 and Sec. 3.3, PDF pp. 6–7. Exact crop from PDF p. 7.

Decay granularity and correction cost shape the design

Operator	Decay	Value factor	Log K/U	M
GDN	Per head	Corrected	BF16 / FP32	8
KDA	Per key	Corrected	BF16 / FP32	4
RWKV6	Per key	Raw value	BF16 / BF16	4

- All dense bases and log-decay metadata use FP32.
- GDN has cheap scalar rescaling; KDA retains elementwise rescaling.
- FP32 corrected values mitigate cancellation along the Delta-rule path.

Paper, Sec. 3.4, PDF p. 7; selected kernel configurations, Table 3, PDF p. 9.

Serving needs phase-aware graphs and state slots

Append queue / graph	Merge queue / graph
Decode through compact appends.	Commit the dense state and clear logs.
Move requests at their merge boundary.	Return requests to the append queue.

- Capture two graph variants per uniform decode-batch descriptor.
- Prefill initializes the base; newly admitted requests start with an empty log.
- Reset auxiliary state when a slot is reused; retain other live slots.

Evaluated in a batch-synchronous, graph-mode vLLM prototype.

Paper, Sec. 4 and implementation scope in Sec. 5.1, PDF pp. 8–9. Diagrammatic layout is a teaching summary.

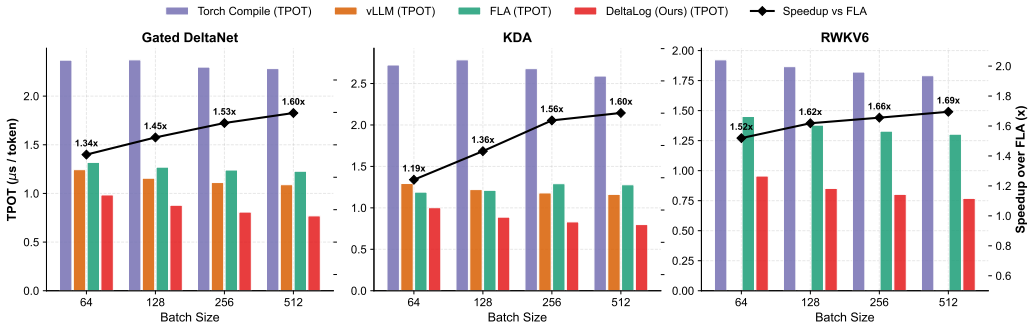
Evaluation separates kernels, mechanism, and serving

Layer	Configuration and comparison
Kernels	H200 NVL / RTX 4090; $H/d_k/d_v = 32/128/128$; batch 64–512; FLA, PyTorch Compile, available vLLM kernels
Mechanism	H200 DRAM counters; split-replay ablation; merge-interval and aggregate-state-size sweeps
Numerics	128-token rollouts; batch 64, 128, 256; output and materialized-state errors against dense references
Serving	H200; Qwen3.6-35B-A3B and Kimi-Linear-48B-A3B-Instruct; 128 generated tokens

Compare complete decode behavior, including periodic merges.

Paper, Secs. 5.1–5.6, PDF pp. 8–12. No prefill performance evaluation.

H200 kernels improve across all three operators

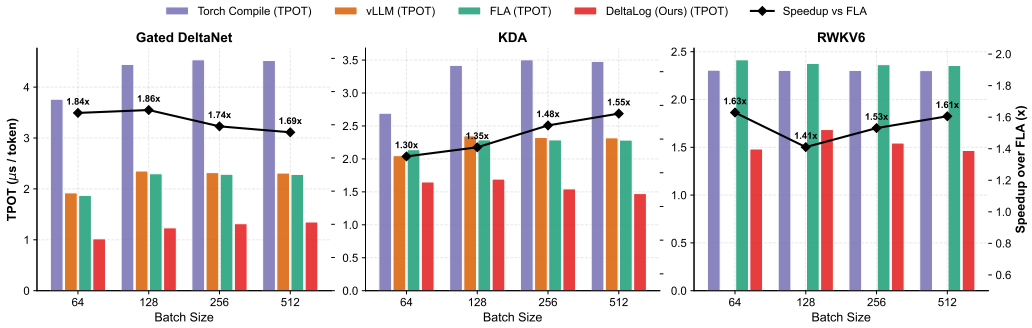


FLA-relative gains: GDN **1.34–1.60**×; KDA **1.19–1.60**×; RWKV6 **1.52–1.69**×.

Bars: reported TPOT on the left axis. Lines: speedup on the right axis.

Paper, Fig. 6, PDF p. 9; Sec. 5.2, PDF pp. 9–10. Original author figure.

RTX 4090 reaches the headline 1.86× kernel gain



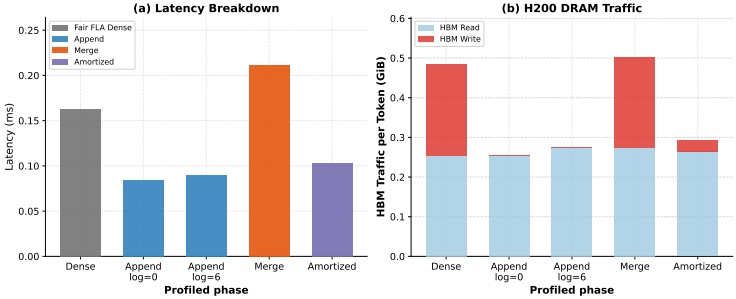
The maximum is GDN at $B = 128$ relative to FLA.

KDA: 1.30–1.55×. RWKV6: 1.41–1.63×.

The dependence on batch size differs from the H200 results.

Paper, Fig. 7, PDF p. 9; Sec. 5.2, PDF p. 10. Original author figure.

Counters confirm fewer writes, with extra replay reads



H200, GDN

$B = 128,\ M = 8$

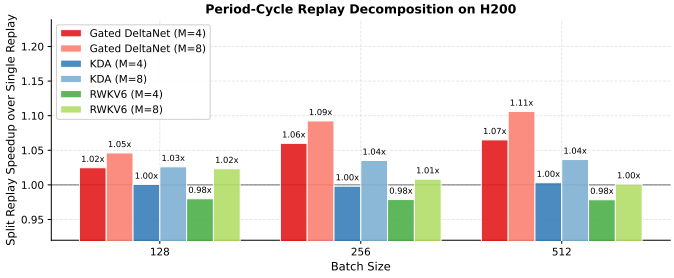
Dense write:
0.230 GiB/token

Amortized write:
0.0294 GiB/token

- Append reads rise from 0.253 to 0.273 GiB as the log grows.
- Reported reduction: **7.83× in writes**, **1.64× in total traffic**.
- Merge restores a dense write, but only once per eight steps.

Paper, Fig. 8 and Sec. 5.3, PDF p. 10. Ratios use the authors' reported precision.

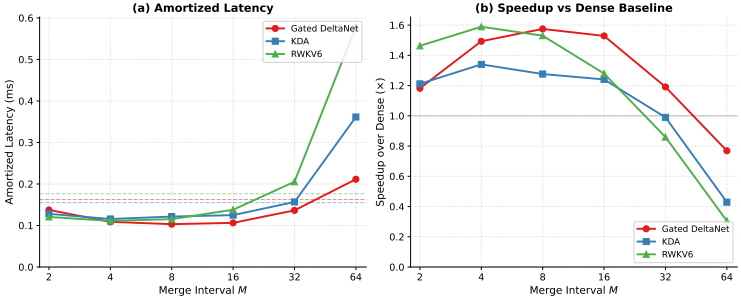
Split replay helps only when reuse pays for preparation



- GDN, $M = 8$: **1.05–1.11**$\times$ faster than fused replay.
- KDA, $M = 4$: essentially neutral (0.998–1.003$\times$).
- RWKV6, $M = 4$: **about 2% slower**; its fused path is already cheap.

Paper, Fig. 9, PDF p. 11; Sec. 5.3, PDF p. 10. Full-cycle CUDA Graph timing, H200.

Longer merge intervals eventually lose to replay

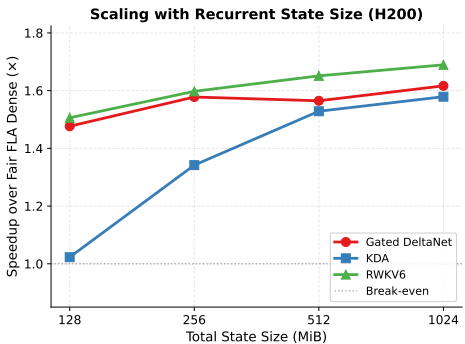


H200, $B = 128$: GDN selects $M = 8$ (0.103 ms);
KDA selects $M = 4$ (0.116 ms); RWKV6 selects $M = 4$ (0.110 ms).

Reducing merge frequency alone is insufficient.

Paper, Fig. 10 and Sec. 5.4, PDF p. 11. Latencies are from this interval sweep.

Larger aggregate states provide more opportunity



128 → 1024 MiB

GDN: 1.48 → 1.62×

KDA: 1.02 → 1.58×

RWKV6: 1.51 → 1.69×

The paper changes batch size, with head dimensions fixed.

KDA nearly breaks even at the smallest state size in this sweep.

Paper, Fig. 11 and Sec. 5.4, PDF p. 11. This campaign differs numerically from Fig. 6; see backup.

Measured equivalence is within floating-point tolerance

Operator	Output relative error (max)	State relative error (max)
GDN	0.00303	7.92×10^{-7}
KDA	0.00450	4.61×10^{-4}
RWKV6	0.00376	2.86×10^{-7}

- 128-token kernel rollouts; $B \in \{64, 128, 256\}$; selected M.
- Authors report passing repository and BF16-aware benchmark tolerances.

What this supports

Mixed precision and reordered sums cause small differences. Agreement is established within tolerance over the measured 128-token horizon.

Paper, Sec. 5.5 and Fig. 12, PDF p. 11. Exact arithmetic is covered separately by the invariant.

Full-model decode gains range from 1.05 to 1.20×

Qwen3.6-35B-A3B / GDN
$M = 8$

B	Dense	DeltaLog	Gain
64	8.36	7.75	1.08×
128	10.30	9.01	1.14×
256	16.07	13.40	1.20×

Kimi-Linear-48B-A3B-Instruct
KDA, $M = 4$

B	Dense	DeltaLog	Gain
64	7.66	7.33	1.05×
128	8.92	8.13	1.10×
256	13.98	12.54	1.12×

TPOT in **ms/token**, lower is better. H200 NVL; 128 generated tokens.

Same models, precision, batch sizes, and graph replay settings.

16.07 → 13.40 ms is a **16.6% latency reduction**.

These measurements cover decode; prefill is excluded.

Paper, Table 4, PDF p. 11; Sec. 5.6, PDF p. 12. Gain values retain paper rounding.

System gains depend on the optimized path's share

$$\mathrm{Speedup}_{\mathrm{model}} \approx \frac{1}{(1-f)+f/s}$$

f: original runtime fraction in the optimized path. s: its speedup.

- MLPs, normalization, scheduling, and other model work remain.
- Extra logs consume memory; merge steps add a periodic cost.
- Smaller states offer less write traffic to recover.

Deployment questions that remain

Long-horizon numerical behavior and tail latency under mixed arrivals are separate from the reported fixed-batch decode measurements.

Paper, Secs. 5.6 and 6, PDF p. 12. Amdahl formula is a teaching expression of the paper's explanation.

The physical state representation is an optimization lever

1. **Preserve the logical recurrence.**
   A dense base and bounded factors can represent the same state.

2. **Optimize the commit schedule.**
   Replace most dense writes with compact appends; tune replay against merge cost.

3. **Measure at the right level.**
   Up to 1.86× kernel speedup; 7.83× fewer profiled writes; 1.05–1.20× full-model decode speedup.

Best supported regime: large-batch recurrent decoding with costly state writes.

Lin, Sun, and Sun. DeltaLog: Deferred Materialization of Recurrent States for Linear Attention Decoding. arXiv:2608.15533v1, 2026. Secs. 6–8, PDF pp. 12–13.

Backup: a scalar example of deferred materialization

Teaching example: $S_0 = 10$, decay $1/2$, updates $2, 3, 4$, $M = 3$.

Step	Logical state	Dense base	Representation
1	7	10	$10/2 + 2$
2	6.5	10	$10/4 + 2/2 + 3$
3	7.25	7.25	$10/8 + 2/4 + 3/2 + 4$

The base remains unchanged for two steps, while the represented logical state changes every step. Step 3 merges and empties the log.

Teaching example, not a paper experiment. Based on Eqs. (6)–(7), PDF pp. 5–6.

Backup: bounded logs add a modest resident footprint

$$C_{\text{entry}} = BH(p_K d_k + p_U d_v + 4 d_\lambda),$$
$$C_{\text{decay}} = 4BHd_\lambda,$$
$$C_{\text{aux}} = (M-1)C_{\text{entry}} + C_{\text{decay}}.$$

p_K, p_U: bytes/element. d_λ: decay width (1 or d_k).

Operator	M	Auxiliary MiB	Relative to base
GDN	8	21.125	8.25%
KDA	4	17.000	6.64%
RWKV6	4	14.000	5.47%

$B = 128$, $H/d_k/d_v = 32/128/128$: FP32 base = 256 MiB.
Excludes scratch, indexing, and alignment; not measured peak memory.

Derived from paper Eq. (9), PDF p. 6, and Table 3, PDF p. 9.

Backup: separate the paper's measurement campaigns

An unresolved numerical difference

Table 3's shape implies $B = 64$ corresponds to 128 MiB of FP32 state. Yet KDA speedup is **1.19× in Fig. 6** and **1.02× in Fig. 11 / Sec. 5.4**. The paper does not explain the difference in settings or timing.

- Report the kernel sweep and state-size sweep separately.
- Use ratios within each campaign; do not merge them into one curve.
- Numerical tests cover 128 steps; the exact relative-error denominator and full tolerance rules are not specified in the text.

Paper, Fig. 6 and Table 3, PDF p. 9; Fig. 11 and Secs. 5.4–5.5, PDF p. 11. Reader comparison.