DISLA: LINEAR ATTENTION WITH DISTINCT USE AND UPDATE DECISIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurrent linear attention stores sequence history in a fixed-size state. Existing gated Delta models use the current input to correct key–value associations in the state, then read the updated state to form an output. Thus, the same correction is committed to the persistent state and contributes to the current output. However, persistent state updates need to account for future outputs, whereas the current output serves only the present step: the two roles may require different state corrections. We introduce DisLA, which learns separate corrections for state updates and current outputs. The two control projections share a base, with a low-rank branch representing their difference. We instantiate this mechanism in Gated DeltaNet-2 (GDN2), preserving its feed-forward width and recurrent state size while supporting chunkwise parallel training. Joint erase–write decoupling in DisLA-GDN2 adds only 0.17% parameters. In matched experiments with approximately 133M parameters and 3B valid next-token prediction targets from FineWeb-Edu per model, DisLA-GDN2 reduces WikiText-2 perplexity by 2.6% and improves average accuracy across LAMBADA and eight reasoning tasks by 1.22 percentage points over GDN2. It also raises the mean score across all 13 RULER tasks at 4K from 1.81 to 2.65.

1 INTRODUCTION

Recurrent linear attention compresses sequence history into a fixed-size state, avoiding a key–value cache that grows with context length during inference (Katharopoulos et al., 2020). This state can be viewed as an associative memory: key–value associations derived from the inputs are continually integrated into a shared state and retrieved through queries (Schlag et al., 2021). Each recurrent step produces two results: an updated state passed to subsequent steps and an output representing the current position. The former maintains memory across steps, while the latter provides the result of the current computation. Both depend on the same input and history, but they serve different temporal roles.

The delta-rule family improves on additive accumulation by making targeted corrections to existing key–value associations (Schlag et al., 2021; Yang et al., 2024b). Subsequent gated models refine this process: Gated DeltaNet combines targeted correction with adaptive forgetting (Yang et al., 2025), and Gated DeltaNet-2 (GDN2) separately controls the erasure of old content and the writing of new values (Hatamizadeh et al., 2026). Despite increasingly flexible update rules, these models retain the same computation pattern: they first update the state and then use the updated state to form the current output. Choosing a correction for the persistent state therefore also determines the correction used in the current output computation. Yet the requirements of these roles may differ: a correction suitable for the persistent state may not be equally suitable for the current output, while a correction needed now may not need to persist in the state. This raises a question distinct from how to improve the update rule: should state updates and current outputs select their corrections separately?

We introduce DisLA, which learns separate corrections for state updates and current outputs. Both paths use the same input and existing state: the update path modifies the persistent state, while the use path supplies the correction contributing to the current output. Because the two roles share this information, we retain a shared base control projection and use a low-rank branch to represent the difference between the two projections. The shared component learns common decisions, while the difference allows distinct control coefficients for the two roles. We instantiate the mechanism as

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.

DisLA-GDN2, which generates separate erase and write gates for each role. It preserves GDN2's feed-forward width and recurrent state size, and supports chunkwise parallel computation of the state and output.

In matched experiments on the GDN2 backbone, DisLA improves language-modeling loss on independently held-out data, average accuracy across LAMBADA and commonsense reasoning tasks, and the mean score on the complete RULER evaluation. Comparisons of write-only, erase-only, and joint decoupling further show that the scope of decoupling affects how gains are distributed across tasks. Context-length evaluations and real-world in-context tasks likewise reveal different strengths among the configurations. These results support modeling state updates and current use separately, while identifying the scope of decoupling as a design choice that affects language quality and in-context performance.

2 PRELIMINARIES

2.1 LINEAR ATTENTION

For a single attention head, let $q_t, k_t \in \mathbb{R}^{d_k}$ and $v_t \in \mathbb{R}^{d_v}$ denote the query, key, and value at position t, where d_k and d_v are the key and value dimensions. We use unnormalized linear attention, with any feature map absorbed into q_t and k_t. The model maintains a state $S_t \in \mathbb{R}^{d_k \times d_v}$, adds the current key–value association through an outer product, and reads the state to produce $o_t \in \mathbb{R}^{d_v}$:

$$S_t = S_{t-1} + k_t v_t^\top, \qquad o_t = S_t^\top q_t. \tag{1}$$

With $S_0 = 0$, expanding the recurrence gives

$$o_t = \sum_{i=1}^{t} (q_t^\top k_i) v_i. \tag{2}$$

Reading the state is therefore equivalent to summing values up to the current position, weighted by query–key inner products, without storing the key–value associations individually (Katharopoulos et al., 2020).

For a sequence of length T, stack the vectors as rows of $Q, K \in \mathbb{R}^{T \times d_k}$ and $V, O \in \mathbb{R}^{T \times d_v}$. Equation (2) then becomes

$$O = \big((QK^\top) \odot M_{\leq}\big)V, \qquad M_{\leq}[t,i] = \mathbf{1}_{i \leq t}. \tag{3}$$

where $\odot$ denotes elementwise multiplication and $M_{\leq}$ is the causal mask including its diagonal. This form explicitly represents causal interactions across positions, but directly constructing $QK^\top$ requires quadratic computation and storage in T for fixed head dimensions.

Chunkwise computation avoids constructing this full-sequence interaction matrix (Yang et al., 2024a). For a chunk of C positions, let S_{in} and S_{out} denote its incoming and outgoing states. Here, Q, K, V, O contain only the current chunk, and $M_{\leq} \in \mathbb{R}^{C \times C}$. The chunk output and state update are

$$\begin{aligned} O &= QS_{\text{in}} + \big((QK^\top) \odot M_{\leq}\big)V, \\ S_{\text{out}} &= S_{\text{in}} + K^\top V. \end{aligned} \tag{4}$$

The first output term reads the history preceding the chunk; the second computes within-chunk causal interactions. The outgoing state accumulates all writes in the chunk and is passed to the next chunk. This formulation combines parallel within-chunk computation with recurrent state propagation, using $C \times C$ interaction matrices. For fixed head dimensions and chunk size, total computation is linear in sequence length.

2.2 DELTA-RULE MEMORY

Additive writes accumulate new key–value associations without accounting for what the state already stores at the current key. The delta rule instead retrieves the value associated with that key and corrects the state using the error between the incoming and retrieved values (Schlag et al., 2021). To

express DeltaNet and its extensions with forgetting in a common form, let $D_t \in \mathbb{R}^{d_k \times d_k}$ denote a decay matrix acting on the previous state and β_t a scalar update coefficient. Then

$$\bar{S}_t = D_t S_{t-1}, \qquad S_t = \bar{S}_t + \beta_t k_t \big(v_t - \bar{S}_t^\top k_t\big)^\top. \tag{5}$$

Here, $\bar{S}_t$ is the decayed state, and $v_t - \bar{S}_t^\top k_t$ is the error between the incoming and retrieved values. The correction is a gradient step of size β_t on the local reconstruction objective $\mathcal{L}_t(S) = \frac{1}{2}\|S^\top k_t - v_t\|_2^2$, starting from $\bar{S}_t$.

DeltaNet uses $D_t = I$, where I is the identity matrix, without an additional decay of the previous state. Gated DeltaNet uses $D_t = \alpha_t I$, where the scalar $\alpha_t \in (0, 1)$ uniformly scales the entire state within each head (Yang et al., 2025).

GDN2 uses channel-wise decay, $D_t = \mathrm{Diag}(\boldsymbol{\alpha}_t)$, where $\boldsymbol{\alpha}_t \in (0, 1)^{d_k}$ contains retention coefficients for individual key channels and Diag constructs a diagonal matrix. It separately controls the erasure of old content and the writing of new values (Hatamizadeh et al., 2026). Let $b_t \in (0, 1)^{d_k}$ and $w_t \in (0, 1)^{d_v}$ denote the erase and write gates. Its update residual is

$$\varepsilon_t = w_t \odot v_t - \bar{S}_t^\top (b_t \odot k_t). \tag{6}$$

The erase gate acts on the key channels used to retrieve old content, while the write gate acts on the incoming value channels. The state update and output are

$$\begin{aligned} S_t &= \bar{S}_t + k_t \varepsilon_t^\top, \\ o_t &= S_t^\top q_t = \bar{S}_t^\top q_t + (k_t^\top q_t)\varepsilon_t. \end{aligned} \tag{7}$$

Setting every channel of both gates to the same scalar β_t recovers Eq. (5); using scalar decay further recovers Gated DeltaNet. Equation (7) shows that the same residual ε_t generates the persistent state correction $k_t \varepsilon_t^\top$ and contributes to the current output after weighting by the query–key inner product $k_t^\top q_t$.

3 DisLA

3.1 Distinct Use and Update Decisions

In Eq. (7), the residual used to update the state also contributes to the current output. DisLA separates these roles by generating a state-update residual $\varepsilon_t^c \in \mathbb{R}^{d_v}$ and a current-use residual $\varepsilon_t^u \in \mathbb{R}^{d_v}$. The superscripts c and u denote *commit* and *use*, respectively. Both residuals are computed from the same current input features and the same decayed state $\bar{S}_t = D_t S_{t-1}$, before either correction is applied. The recurrence and output are

$$\begin{aligned} S_t &= \bar{S}_t + k_t (\varepsilon_t^c)^\top, \\ o_t &= \bar{S}_t^\top q_t + (k_t^\top q_t)\varepsilon_t^u. \end{aligned} \tag{8}$$

The commit residual ε_t^c corrects the state along the current key k_t, producing S_t for the next step. The use residual ε_t^u, weighted by the query–key inner product $k_t^\top q_t$, combines with the read from $\bar{S}_t$ to form the current output.

Substituting the state update in Eq. (8) into the output expression gives

$$o_t = S_t^\top q_t + (k_t^\top q_t)(\varepsilon_t^u - \varepsilon_t^c). \tag{9}$$

Thus, the output is equivalent to reading the updated state and correcting the current update's contribution according to the difference between the two residuals. When $\varepsilon_t^u = \varepsilon_t^c$, the additional term vanishes, recovering the post-update readout $o_t = S_t^\top q_t$. Otherwise, the output can use a different correction from the one committed to memory, without maintaining a second recurrent state. The two computations share queries, keys, values, and decay; their difference is controlled by the projections that generate their control coefficients.

3.2 Shared Projections with Low-Rank Differences

The two roles use the same input features but need not select the same control coefficients. We retain a shared base projection and represent their difference with a low-rank branch, separating current

use from persistent updates with few additional parameters. Let $h_t \in \mathbb{R}^d$ be the normalized layer input and m the number of control channels across all heads. With $W_0 \in \mathbb{R}^{m\times d}$, $A \in \mathbb{R}^{r\times d}$, and $B \in \mathbb{R}^{m\times r}$, the commit and use logits are

$$\ell_t^c = W_0 h_t - BAh_t, \qquad \ell_t^u = W_0 h_t + BAh_t. \tag{10}$$

The base supplies a common control signal; the branch BAh_t shifts the two roles in opposite directions. In DisLA-GDN2, an elementwise sigmoid maps these logits to gates, which are partitioned by head and by erase or write function.

The effective projections $W_c = W_0 - BA$ and $W_u = W_0 + BA$ share the base matrix W_0, with the trainable factors A and B generating opposite offsets. Their difference is $W_u - W_c = 2BA$. We use an intermediate dimension of $r = 8$, with one factor pair spanning the selected control channels of all heads in each layer.

Following the zero-output initialization used in LoRA (Hu et al., 2022), we initialize A randomly and B to zero. Both projections initially equal W_0, recovering the GDN2 recurrence and readout when common parameters and initial states match. We train W_0, A, and B jointly under the same language-modeling loss, allowing the two roles to learn different controls from this shared starting point. Appendix A.1 gives the factor gradients and the equivalence at initialization.

3.3 GDN2 Instantiation and Architecture

For each role $s \in \{c, u\}$, the GDN2 instance generates an erase gate $b_t^s \in (0,1)^{d_k}$ and a write gate $w_t^s \in (0,1)^{d_v}$. Define

$$e_t^s = b_t^s \odot k_t, \qquad z_t^s = w_t^s \odot v_t, \qquad \varepsilon_t^s = z_t^s - \bar{S}_t^\top e_t^s. \tag{11}$$

The erase gate selects the old content to subtract, and the write gate selects the incoming value to insert. Substituting the two residuals into Eq. (8) lets the current output use a different edit from the one retained in memory. Both roles share the key, value, and decay, as well as the state before correction.

Joint decoupling applies Eq. (10) to both gate groups. Write-only decoupling keeps $b_t^c = b_t^u$ and separates only the write projections; erase-only keeps $w_t^c = w_t^u$ and separates only the erase projections. These variants change the scope of the difference branch without changing the backbone or state dimensions.

The two gate groups act on different terms in the role difference:

$$\varepsilon_t^u - \varepsilon_t^c = (w_t^u - w_t^c) \odot v_t - \bar{S}_t^\top \big((b_t^u - b_t^c) \odot k_t\big). \tag{12}$$

Write-only decoupling changes the contribution of the incoming value; erase-only changes how much old content is subtracted along the current key. Joint decoupling learns both terms. Equation (9) maps this residual difference to the output through the shared query–key inner product.

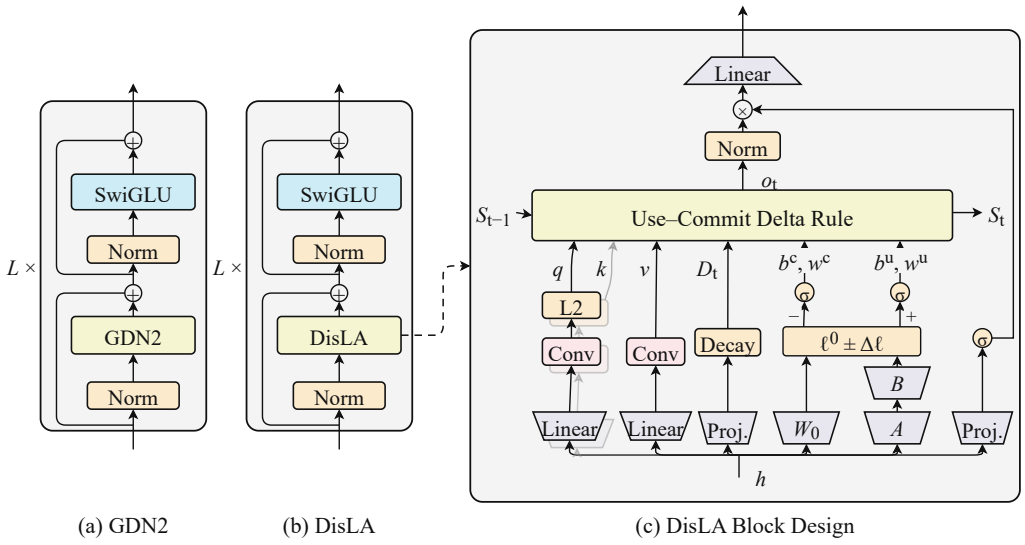
Figure 1 shows the complete layer. GDN2 and DisLA share pre-normalization, residual connections, and SwiGLU (Shazeer, 2020). Query, key, and value projections pass through short convolutions with SiLU; queries and keys are then L2-normalized. For this instance, q_t in our equations includes the subsequent $d_k^{-1/2}$ query scaling. The role gates use the normalized layer input. Head-wise RMS normalization (Zhang & Sennrich, 2019), a sigmoid output gate, and a linear projection transform the operator output before the residual addition, using the FLA layer conventions (Yang & Zhang, 2024).

3.4 Chunkwise Computation and Complexity

DisLA retains GDN2's chunkwise state computation: commit controls the updates, and use reads the resulting history. Consider a block of C tokens with incoming state $S_{\text{in}} \in \mathbb{R}^{d_k\times d_v}$. Before each current edit, memory contains the decayed incoming state and earlier commits within the block.

Stack the commit residuals as rows, $R^c[t,:] = (\varepsilon_t^c)^\top$, so $R^c \in \mathbb{R}^{C\times d_v}$. Each residual is the write target minus the read from old memory, giving

$$R^c = Z_c - F_c S_{\text{in}} - L_c R^c. \tag{13}$$



(a) GDN2 (b) DisLA (c) DisLA Block Design

Figure 1: **GDN2 and DisLA architectures.** (a,b) The models share the residual backbone and feed-forward width. (c) The expanded mixer generates commit ($-$) and use ($+$) gates from a shared projection and a low-rank difference. Commit updates the persistent state from S_{t-1} to S_t; use supplies the current output o_t. The dashed arrow identifies the expanded mixer. Here $\ell^0 = W_0 h_t$ and $\Delta\ell = BAh_t$; the diagram suppresses time indices, and L denotes the number of layers.

Here $Z_c \in \mathbb{R}^{C\times d_v}$ collects commit-gated write targets. The coefficients $F_c \in \mathbb{R}^{C\times d_k}$ read the decayed incoming state, while $L_c \in \mathbb{R}^{C\times C}$ accounts for earlier commits read along the current erase direction. The two subtracted terms therefore represent history from before and within the block, respectively.

Only earlier positions contribute to each residual, making L_c strictly lower triangular. Moving the residual terms to the left gives

$$(I_C + L_c)R^c = Z_c - F_c S_{\text{in}}. \tag{14}$$

The identity retains each position's own residual; left multiplication by L_c combines earlier residual rows. Keys, commit gates, and decay determine the coefficients, preserving GDN2's chunkwise triangular solve (Hatamizadeh et al., 2026).

For the output, write $\kappa_t = k_t^\top q_t$. Substituting the use residual and collecting the old-state terms gives

$$\begin{aligned} o_t &= \bar{S}_t^\top q_t + \kappa_t(z_t^u - \bar{S}_t^\top e_t^u) \\ &= \bar{S}_t^\top \widetilde{q}_t + \kappa_t z_t^u, \qquad \widetilde{q}_t = q_t - \kappa_t e_t^u. \end{aligned} \tag{15}$$

The adjusted query incorporates use-erase into the historical read; the second term supplies the current use-gated value. Let $\text{Diag}(\gamma_t) = D_t \cdots D_1$ denote cumulative decay. The block output $O \in \mathbb{R}^{C\times d_v}$ and outgoing state are

$$O = \widetilde{Q}_\gamma S_{\text{in}} + \widetilde{P}_< R^c + \text{Diag}(\boldsymbol{\kappa}) Z_u, \tag{16}$$

$$S_{\text{out}} = \text{Diag}(\gamma_C) S_{\text{in}} + K_{\text{tail}}^\top R^c. \tag{17}$$

The output terms read incoming memory through $\widetilde{Q}_\gamma$, read earlier commits through $\widetilde{P}_<$, and add current use-gated values Z_u, weighted by the inner products in $\boldsymbol{\kappa}$. The outgoing state decays incoming memory and all commits to the block end, using the weighted keys K_{tail}. Appendix A.2 defines the coefficients and derives both expressions.

The low-rank projection adds $r(d+m)$ parameters per layer and $O(r(d+m))$ operations per token per layer. For both gate groups in GDN2 with H heads, $m = H(d_k + d_v)$. The persistent matrix remains $d_k \times d_v$ per head, with unchanged short-convolution caches. For fixed block size and model dimensions, the update and readout costs remain linear in sequence length.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We compare four configurations on the GDN2 backbone: GDN2, DisLA-GDN2 with use–commit decoupling in both gate groups, and its write-only and erase-only variants. All four have 12 layers, hidden width 768, 12 heads of dimension 64, tied input and output embeddings, and SwiGLU width 1792. They retain 589,824 matrix-state scalars per sequence. The joint rank-8 branch increases the parameter count from 133,154,448 to 133,375,632; each single-scope variant has 133,301,904 parameters. All common parameters have paired initialization. All four models use a shared chunk implementation, with use and commit gates tied for GDN2. Unless qualified, DisLA denotes the joint configuration in pairwise comparisons below.

Each model is pretrained on 3B next-token prediction targets from FineWeb-Edu (Penedo et al., 2024), using the GPT-NeoX tokenizer (Black et al., 2022) and independent document windows of at most 4096 input tokens. We use AdamW (Loshchilov & Hutter, 2019), 32,768 valid targets per update, 30M targets of warmup, and cosine decay to 10% of the peak learning rate. For the parent–joint comparison, each model receives three 100M-target trials at peak rates $\{7.5 \times 10^{-4}, 1.5 \times 10^{-3}, 3 \times 10^{-3}\}$; both select 1.5×10^{-3} by terminal validation loss. All four main runs use this peak rate and seed 123, selecting the lowest-loss checkpoint among ten scheduled validation evaluations. Appendices B.1 and C give the recipe and learning-rate search results.

Language evaluation uses independent FineWeb confirmation documents, WikiText-2 (Merity et al., 2016), LAMBADA (Paperno et al., 2016), and eight reasoning tasks through the LM Evaluation Harness (Gao et al., 2023). These likelihood-based tasks are zero-shot. In-context evaluation uses RULER's native prompts and examples (Hsieh et al., 2024) and six real-world tasks adapted from the Based and JRT evaluation suites (Arora et al., 2024a;b). We report pointwise paired bootstrap intervals over evaluation documents or examples in Appendix C. These intervals quantify uncertainty over evaluation samples for the trained parent–joint pair.

4.2 LANGUAGE MODELING AND COMMONSENSE REASONING

All three scopes improve independent FineWeb NLL without enlarging persistent memory (Table 1). Joint leads, lowering NLL from 3.1596 to 3.1546 with a paired 95% interval of $[-0.0057, -0.0044]$, and WikiText-2 word perplexity from 50.24 to 48.92, a 2.6% reduction. Its advantage at all ten development-validation checkpoints carries over to independent confirmation.

Joint and write-only both reach 40.94% nine-task accuracy, up 1.22 points over GDN2. BoolQ contributes most of the joint gain at 9.51 points; WinoGrande and OpenBookQA improve by 2.13 and 1.00 points. The two configurations reach this mean through different task profiles. Write-only yields the lowest LAMBADA perplexity and highest BoolQ accuracy; its BoolQ score exceeds Joint by 1.80 points. Erase-only leads HellaSwag, WinoGrande, and ARC-Easy, with gains of 0.59, 3.31, and 1.73 points over GDN2. It achieves these gains while separating only the control over old-content subtraction. Appendix C gives training trajectories and paired intervals.

4.3 IN-CONTEXT EVALUATION

Joint leads RULER at 4K, raising the 13-task mean from 1.81% to 2.65% (Table 2). Aggregation and QA drive the gain: joint leads all four tasks, including frequent-word extraction at 10.93% versus 5.87%. Needle retrieval favors other scopes: write-only leads S-NIAH-2 and MK-NIAH-1, and erase-only leads S-NIAH-1.

Figure 2 tracks eight fixed needle tasks from 1K to 8K. Write-only leads S-NIAH-2 and MK-NIAH-1 at 2K, 4K, and 8K. Joint improves S-NIAH-2 over GDN2 at every length; erase-only leads S-NIAH-1 at 4K and 8K. The leading configurations at 1K are different: erase-only leads S-NIAH-2, and GDN2 leads MK-NIAH-1. Write-only overtakes both from 2K onward, so the short-context ranking does not determine the ranking at longer lengths.

All scopes improve the real-world mean, led by write-only at 13.98% versus GDN2's 13.20% (Table 3). Its lead spans SWDE, SQuAD completion, FDA, TriviaQA, and NQ. Joint leads DROP at 11.93% versus 10.54%. The ordering reverses even within question answering: Joint leads both

Table 1: **Language modeling and zero-shot reasoning on the GDN2 backbone.** All configurations use 3B training targets and validation-selected checkpoints. Joint denotes DisLA-GDN2 with use–commit decoupling in both erase and write gates.

Metric	GDN2	Joint	Write-only	Erase-only
FineWeb NLL ↓	3.1596	**3.1546**	3.1556	3.1558
WikiText-2 PPL ↓	50.24	**48.92**	50.71	49.93
LAMBADA PPL ↓	126.63	129.40	**126.52**	132.11
LAMBADA acc.	**23.17**	22.78	**23.17**	23.04
PIQA	**62.57**	62.13	62.30	62.35
HellaSwag	32.30	32.56	32.70	**32.89**
WinoGrande	50.04	52.17	50.59	**53.35**
ARC-Easy	50.76	50.84	51.81	**52.48**
ARC-Challenge	25.34	24.49	**25.43**	24.57
OpenBookQA	29.80	**30.80**	28.20	30.00
Social IQa	**37.92**	37.62	37.41	36.75
BoolQ	45.57	55.08	**56.88**	50.73
Avg.9	39.72	**40.94**	**40.94**	40.69

Joint, write-only, and erase-only denote DisLA-GDN2 variants. Accuracies are percentages; HellaSwag, ARC-Challenge, and OpenBookQA use normalized accuracy. Avg.9 equally weights LAMBADA and the eight reasoning tasks. FineWeb uses the independent confirmation set; WikiText-2 reports word perplexity.

Table 2: **Complete RULER evaluation at 4K.** All four configurations use the GDN2 backbone. Each task uses 500 examples. Scores are percentages; Avg.13 averages the 13 tasks rather than the four family means.

Family	Task	GDN2	Joint	Write-only	Erase-only
Retrieval	S-1	0.80	0.00	0.20	**1.20**
	S-2	0.60	3.80	**4.00**	0.20
	S-3	0.00	0.00	0.00	0.00
	MK-1	3.20	2.80	**4.20**	2.60
	MK-2	0.00	0.00	0.00	0.00
	MK-3	0.00	0.00	0.00	0.00
	MQ	**0.50**	0.05	0.20	0.05
	MV	**0.45**	0.05	0.00	0.05
Tracking	VT	0.00	0.00	0.00	0.00
Aggregation	CWE	0.26	**0.88**	0.46	0.12
	FWE	5.87	**10.93**	4.07	4.07
QA	QA-1	5.20	**6.80**	5.00	6.00
	QA-2	6.60	**9.20**	9.00	7.60
Avg.13		1.81	**2.65**	2.09	1.68

S/MK/MQ/MV: single-needle, multi-key, multi-query, multi-value; VT: variable tracking; CWE/FWE: common-/frequent-word extraction.

RULER QA tasks, whereas write-only leads TriviaQA, NQ, and SQuAD completion. The best scope on the synthetic suite therefore differs from that on natural-document questions. Appendix B.3 details the tasks; Appendix C gives intervals and output diagnostics.

4.4 Decoupling Scope

The single-scope variants isolate the two terms of Eq. (12): write-only separates controls on incoming values while sharing erase gates; erase-only separates old-content erasure while sharing write

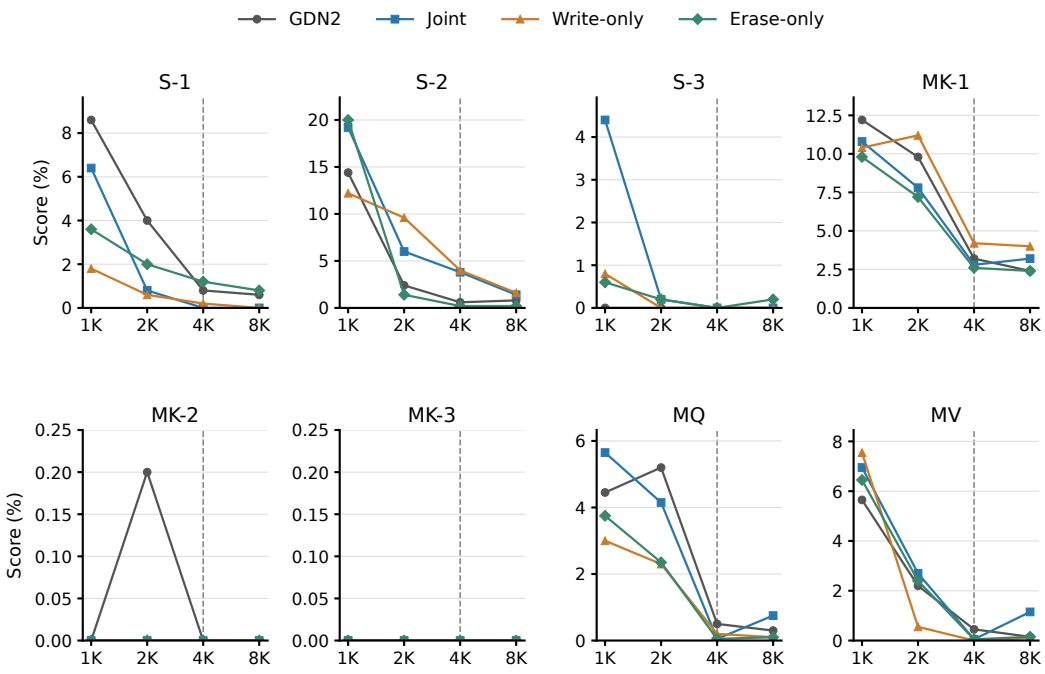


Figure 2: **Needle scores across context length.** The four configurations use a fixed eight-task subset with 500 examples per condition. The vertical dashed line marks the 4K training boundary; 8K is length extrapolation. Scores can include partial credit for tasks with multiple reference strings. Axes use task-specific vertical ranges; exact values appear in Appendix C.

Table 3: **Six real-world in-context tasks at 2K.** The four GDN2 configurations use the same adapted task sets and case-insensitive reference-answer containment scoring.

Model	SWDE	SQuAD-c	FDA	TriviaQA	NQ	DROP	Avg.6
GDN2	7.92	18.80	1.54	33.89	6.49	10.54	13.20
Joint	4.95	19.87	1.63	35.01	6.49	**11.93**	13.32
Write-only	**8.46**	**20.44**	**2.09**	**35.72**	**6.94**	10.21	**13.98**
Erase-only	5.94	19.47	1.72	35.31	6.87	11.26	13.43

SQuAD-c denotes SQuAD completion. Scores are answer-containment percentages; Avg.6 equally weights the six tasks.

gates. Joint learns both role differences. These comparisons ask whether gains require changing both parts of a memory edit. Write-only shares erase control between use and commit, yet improves the real-world mean and matches Joint on Avg.9. Conversely, erase-only improves FineWeb NLL and several reasoning tasks while keeping the write projection shared across roles. Separating use from commit in either term can therefore be useful, with the term selected affecting where the gains appear.

All preserve the backbone and persistent-state capacity. Single-scope branches add 147,456 parameters each, versus 221,184 for joint decoupling. Write-only's lead on five real-world tasks and their mean therefore makes role-specific value control alone a competitive, smaller configuration; joint achieves the lowest FineWeb NLL and WikiText-2 perplexity, and the highest mean RULER score.

4.5 Efficiency

All three decoupling scopes retain the same recurrent matrices and short-convolution buffers as GDN2, giving a full decoding cache of 2520 KiB per sequence (Table 4). Joint decoupling adds

Table 4: **Efficiency and persistent storage on an RTX 4090 D.** Cache size includes recurrent matrices and short-convolution states.

Model	Training targets/s↑	Prefill@4K tokens/s↑	Decode tokens/s↑	Peak train GiB↓	Full cache KiB/seq.↓
GDN2	**26343**	**109246**	**32.45**	**9.78**	2520
Joint	24427	102743	30.72	10.14	2520
Write-only	25157	103728	30.38	10.14	2520
Erase-only	25060	103653	31.18	10.14	2520

Training and prefill use freshly initialized models; decode uses the selected checkpoints with BF16 autocast.

0.166% parameters. On an RTX 4090 D, its training, prefill, and decoding throughput are 7.3%, 6.0%, and 5.4% lower than GDN2, respectively, while peak training memory rises from 9.78 to 10.14 GiB.

Training on 3B prediction targets took 33.9 hours for GDN2 and 35.7 hours for joint DisLA, including validation and checkpoint saving. The throughput measurements use short runs; Appendix D gives the timing and cache-accounting procedures.

5 Related Work

Recurrent sequence models. Linear attention links attention-like computation to finite recurrent states (Katharopoulos et al., 2020), including a fast-weight interpretation of associative memory (Schlag et al., 2021). DeltaNet and GDN improve how those associations are corrected and forgotten (Yang et al., 2024b; 2025). MesaNet instead studies memory updates through an in-context regression objective and analyzes position-dependent prediction quality (von Oswald et al., 2025). Mamba-2 develops the connection between attention and state-space models, while Mamba-3 advances state-space discretization and transitions (Dao & Gu, 2024; Lahoti et al., 2026). Our focus is the coupling between the edit retained in this state and the edit used for the current output.

Use and memory updates. Different read and update paths also appear in SILA, which combines a pre-update state read with a gated current-value path and a state-conditioned memory update (Zhang et al., 2025). Its training includes a weighting scheme informed by reference attention and distance. GDN2 separates erase and write within a single edit (Hatamizadeh et al., 2026). DisLA assigns both operations role-specific gates, computes the two edits from the same decayed state, and learns the difference between their control projections through a low-rank branch under ordinary next-token training.

Low-rank parameterization. LoRA represents a weight adjustment by two low-rank factors around a frozen pretrained base (Hu et al., 2022). We use its zero-output initialization, but train the base projection jointly from the start. The factors parameterize the difference between use and update projections. In the joint configuration, one factor pair spans the erase and write channels of all heads in a layer.

6 Conclusion

DisLA computes distinct corrections for the current output and persistent memory from shared input features and the same decayed state. A shared projection retains their common decision, while a low-rank branch learns their difference without adding recurrent state. On the studied GDN2 backbone, this change improves language-modeling loss and average downstream accuracy with 0.17% additional parameters. The component comparisons show that the scope of decoupling matters: joint control gives the strongest RULER aggregate, while write-only gives the strongest mean on the real-world tasks. The experiments cover one model scale; broader scaling behavior remains an open question.

AI USE STATEMENT

Generative AI tools assisted with manuscript preparation and editing. The author takes responsibility for the final content.

ETHICS STATEMENT

The study uses publicly released web-derived training data and public evaluation datasets. Such data can contain personal information, harmful content, and social biases despite filtering. Our models are research artifacts for studying recurrent architectures; this work does not evaluate their safety for deployment.

REPRODUCIBILITY STATEMENT

Appendix A gives the parameterization properties and chunkwise derivation. Appendix B specifies model shapes, initialization, optimization, data processing, checkpoint selection, and evaluation metrics. Appendix C reports task-level results and paired evaluation intervals. Code, configurations, and result tables are available at `https://anonymous.4open.science/r/disla-review-F811/`.

REFERENCES

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff, 2024a. URL https://arxiv.org/abs/2402.18668.

Simran Arora, Aman Timalsina, Aaryan Singhal, Benjamin Spector, Sabri Eyuboglu, Xinyi Zhao, Ashish Rao, Atri Rudra, and Christopher Ré. Just read twice: closing the recall gap for recurrent language models, 2024b. URL https://arxiv.org/abs/2407.05483.

Christian Bischof and Charles Van Loan. The WY representation for products of Householder matrices. *SIAM Journal on Scientific and Statistical Computing*, 8(1):s2–s13, 1987. doi: 10.1137/0908009. URL https://doi.org/10.1137/0908009.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about Physical Commonsense in Natural Language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. URL https://arxiv.org/abs/1911.11641.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. GPT-NeoX-20B: An Open-Source Autoregressive Language Model, 2022. URL https://arxiv.org/abs/2204.06745.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, 2019. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

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, 2018. URL https://arxiv.org/abs/1803.05457.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 10041–10071, 2024. URL https://proceedings.mlr.press/v235/dao24a.html.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, 2019. doi: 10.18653/v1/N19-1246. URL https://aclanthology.org/N19-1246/.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

Ali Hatamizadeh, Yejin Choi, and Jan Kautz. Gated DeltaNet-2: Decoupling erase and write in linear attention, 2026. URL https://arxiv.org/abs/2605.22791v1.

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/2404.06654v3.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp. 5156–5165, 2020. URL https://proceedings.mlr.press/v119/katharopoulos20a.html.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

Aakash Lahoti, Kevin Y. Li, Berlin Chen, Caitlin Wang, Aviv Bick, J. Zico Kolter, Tri Dao, and Albert Gu. Mamba-3: Improved sequence modeling using state space principles. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8abd2043b71a074278d5f687947bff9c-Abstract-Conference.html.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1711.05101.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models, 2016. URL https://arxiv.org/abs/1609.07843.

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. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

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. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.nips.cc/paper_files/paper/2024/hash/370df50ccfdf8bde18f8f9c2d9151bda-Abstract-Datasets_and_Benchmarks_Track.html. Datasets and Benchmarks Track.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, 2016. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264/.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 784–789, Melbourne, Australia, jul 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2124. URL https://aclanthology.org/P18-2124/.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An Adversarial Winograd Schema Challenge at Scale, 2019. URL https://arxiv.org/abs/1907.10641.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473, 2019. doi: 10.18653/v1/D19-1454. URL https://aclanthology.org/D19-1454/.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pp. 9355–9366, 2021. URL https://proceedings.mlr.press/v139/schlag21a.html.

Noam Shazeer. GLU Variants Improve Transformer, 2020. URL https://arxiv.org/abs/2002.05202.

Johannes von Oswald, Nino Scherrer, Seijin Kobayashi, Luca Versari, Songlin Yang, Sarthak Mittal, Maximilian Schlegel, Kaitlin Maile, Yanick Schimpf, Oliver Sieberling, Alexander Meulemans, Rif A. Saurous, Guillaume Lajoie, Charlotte Frenkel, Razvan Pascanu, Blaise Agüera y Arcas, and João Sacramento. MesaNet: Sequence modeling by locally optimal test-time training, 2025. URL https://arxiv.org/abs/2506.05233v2.

Songlin Yang and Yu Zhang. FLA: A Triton-based library for hardware-efficient implementations of linear attention mechanism, 2024. URL https://github.com/fla-org/flash-linear-attention.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 56501–56523. PMLR, 2024a. URL https://proceedings.mlr.press/v235/yang24ab.html.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, volume 37, 2024b. doi: 10.52202/079017-3668. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/d13a3eae72366e61dfdc7eea82eeb685-Abstract-Conference.html.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/4904fad153f6434a7bcf04465d4be2cc-Abstract-Conference.html.

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. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Biao Zhang and Rico Sennrich. Root Mean Square Layer Normalization. In *Advances in Neural Information Processing Systems*, 2019. URL https://arxiv.org/abs/1910.07467.

Jiayu Zhang, Shuyang Chen, Qian Zheng, Rui Yan, Gang Pan, and Huajin Tang. SILA: Enhancing long-context retrieval capability of linear attention via selective ignoring, 2025. URL https://openreview.net/forum?id=csZndfXLpC. Manuscript.

A Derivations and Algorithms

A.1 Properties of the Parameterization

Parent endpoint. When $BA = 0$, the use and commit logits coincide, so $\varepsilon_t^u = \varepsilon_t^c$. The output is

$$\begin{aligned} o_t &= \bar{S}_t^\top q_t + (k_t^\top q_t)\varepsilon_t^c \\ &= \left(\bar{S}_t + k_t(\varepsilon_t^c)^\top\right)^\top q_t = S_t^\top q_t. \end{aligned} \tag{18}$$

With matching inputs, common parameters, and initial states, the commit path performs the GDN2 update and the expression above recovers its post-update readout. Applying this relation inductively gives identical states and outputs at every token, including for nonzero initial states. Matching the remaining layer parameters extends the equivalence to the full network in exact arithmetic. Different accumulation orders and finite-precision kernels can introduce numerical differences.

The rank bound follows from

$$W_u - W_c = 2BA, \qquad \operatorname{rank}(W_u - W_c) \le \min\{\operatorname{rank}(B), \operatorname{rank}(A)\} \le r. \tag{19}$$

It applies to the linear projection difference before the sigmoid, rather than to gate differences or recurrent states.

Shared and differential gradients. Let $\mathcal{L}$ be the model's language-modeling loss. For the layer under consideration, write it as $\mathcal{L}(W_c, W_u, \theta)$, where θ denotes the remaining parameters. Holding θ fixed, define the gradients of this same loss with respect to the two effective projections as

$$G_c = \nabla_{W_c}\mathcal{L}, \qquad G_u = \nabla_{W_u}\mathcal{L}, \qquad G_c, G_u \in \mathbb{R}^{m\times d}. \tag{20}$$

With $\langle X, Y\rangle_F = \sum_{i,j} X_{ij}Y_{ij}$ denoting the Frobenius inner product, the loss differential for changes in these two projections is

$$\mathrm{d}\mathcal{L} = \langle G_c, \mathrm{d}W_c\rangle_F + \langle G_u, \mathrm{d}W_u\rangle_F. \tag{21}$$

Let $\Delta W = BA$. From $W_c = W_0 - \Delta W$ and $W_u = W_0 + \Delta W$,

$$\mathrm{d}W_c = \mathrm{d}W_0 - \mathrm{d}(\Delta W), \qquad \mathrm{d}W_u = \mathrm{d}W_0 + \mathrm{d}(\Delta W). \tag{22}$$

Substitution and collection of terms give

$$\mathrm{d}\mathcal{L} = \langle G_c + G_u, \mathrm{d}W_0\rangle_F + \langle G_u - G_c, \mathrm{d}(\Delta W)\rangle_F, \tag{23}$$

so

$$\nabla_{W_0}\mathcal{L} = G_c + G_u, \qquad \nabla_{\Delta W}\mathcal{L} = G_u - G_c. \tag{24}$$

A change to the shared projection acts in the same direction on both roles and receives the sum of their gradients. A change to the difference matrix acts in opposite directions and receives their difference.

Writing $G_\Delta = G_u - G_c$ and using $\mathrm{d}(\Delta W) = (\mathrm{d}B)A + B(\mathrm{d}A)$ yields

$$\begin{aligned}\langle G_\Delta, \mathrm{d}(\Delta W)\rangle_F &= \langle G_\Delta, (\mathrm{d}B)A + B(\mathrm{d}A)\rangle_F \\ &= \langle G_\Delta A^\top, \mathrm{d}B\rangle_F + \langle B^\top G_\Delta, \mathrm{d}A\rangle_F.\end{aligned} \tag{25}$$

The factor gradients are therefore

$$\nabla_B \mathcal{L} = G_\Delta A^\top, \qquad \nabla_A \mathcal{L} = B^\top G_\Delta. \tag{26}$$

Here ΔW is an intermediate quantity in the chain rule; the optimizer updates W_0, A, and B.

We sample $A_{ij} \sim \mathcal{N}(0, 1/d)$ and set $B = 0$. Thus $\nabla_A \mathcal{L} = 0$ at initialization, whereas $\nabla_B \mathcal{L} = G_\Delta A^\top$ can be nonzero. The two roles start with identical controls while the difference branch can begin learning. At the tied endpoint with matching common parameters, $G_c + G_u$ equals the corresponding parent gate-projection gradient. Subsequent updates learn shared controls through W_0 and the projection difference through A and B.

A.2 Chunkwise Computation

We work with one head and index positions within a block from 1 to C, with incoming state S_{in}. The commit recurrence is

$$\bar{S}_t = D_t S_{t-1}, \qquad S_t = \bar{S}_t + k_t(\varepsilon_t^c)^\top, \qquad \varepsilon_t^c = z_t^c - \bar{S}_t^\top e_t^c. \tag{27}$$

Cumulative decay and history. Let $g_t \in \mathbb{R}^{d_k}$ be the log-decay vector and define

$$D_t = \mathrm{Diag}(\exp(g_t)), \qquad G_0 = 0, \qquad G_t = \sum_{j=1}^{t} g_j, \qquad \gamma_t = \exp(G_t). \tag{28}$$

Exponentials are elementwise. An edit inserted at i undergoes the subsequent decays

$$D_{t:i} = D_t D_{t-1} \cdots D_{i+1} = \mathrm{Diag}\big(\exp(G_t - G_i)\big), \qquad 0 \le i \le t, \tag{29}$$

with $D_{t:t} = I_{d_k}$. Incoming memory thus carries $D_{t:0}$, while a commit at i carries only $D_{t:i}$. Expanding the recurrence before the current edit gives

$$\bar{S}_t = D_{t:0} S_{\mathrm{in}} + \sum_{i<t} D_{t:i} k_i (\varepsilon_i^c)^\top. \tag{30}$$

The two terms contain incoming memory and earlier commits within the block.

Triangular system for commit residuals. Substituting Eq. (30) into the residual definition yields

$$\varepsilon_t^c = z_t^c - S_{\mathrm{in}}^\top D_{t:0} e_t^c - \sum_{i<t} \big((e_t^c)^\top D_{t:i} k_i\big)\varepsilon_i^c. \tag{31}$$

The two subtractions remove reads from incoming memory and earlier commits along the current erase direction. Their coefficients already include the appropriate decay. Define

$$R^c[t,:] = (\varepsilon_t^c)^\top, \qquad Z_c[t,:] = (z_t^c)^\top, \qquad F_c[t,:] = (e_t^c)^\top D_{t:0} = (\gamma_t \odot e_t^c)^\top, \tag{32}$$

$$(L_c)_{ti} = \begin{cases} (e_t^c)^\top D_{t:i} k_i, & i < t, \\ 0, & i \ge t. \end{cases} \tag{33}$$

Here $R^c, Z_c \in \mathbb{R}^{C \times d_v}$, $F_c \in \mathbb{R}^{C \times d_k}$, and $L_c \in \mathbb{R}^{C \times C}$. In particular,

$$(L_c R^c)[t,:] = \sum_{i<t} (L_c)_{ti} (\varepsilon_i^c)^\top. \tag{34}$$

Transposing each residual equation and stacking the rows gives $R^c = Z_c - F_c S_{\mathrm{in}} - L_c R^c$, hence

$$(I_C + L_c) R^c = Z_c - F_c S_{\mathrm{in}}. \tag{35}$$

Since L_c is strictly lower triangular, $I_C + L_c$ has unit diagonal and the system has a unique solution. With

$$J_c = (I_C + L_c)^{-1}, \qquad U_c = J_c Z_c, \qquad Y_c = J_c F_c, \tag{36}$$

we obtain $R^c = U_c - Y_c S_{\mathrm{in}}$. The auxiliaries U_c, Y_c depend on block inputs; incoming memory enters through this final expression. This follows the WY/UT organization of DeltaNet and GDN2 (Yang et al., 2024b; Hatamizadeh et al., 2026), based on matrix representations of transformation products (Bischof & Van Loan, 1987). The implementation forms the auxiliaries with triangular-transform kernels and then propagates states across blocks.

Use readout and outgoing state. Using $\widetilde{q}_t = q_t - \kappa_t e_t^u$ and $\kappa_t = k_t^\top q_t$, Eq. (15) reads $o_t = S_t^\top \widetilde{q}_t + \kappa_t z_t^u$. Substituting Eq. (30) and transposing gives

$$o_t^\top = \widetilde{q}_t^\top D_{t:0} S_{\text{in}} + \sum_{i<t} \big(\widetilde{q}_t^\top D_{t:i} k_i\big)(\varepsilon_i^c)^\top + \kappa_t (z_t^u)^\top. \tag{37}$$

Define

$$\widetilde{Q}_\gamma[t,:] = (\gamma_t \odot \widetilde{q}_t)^\top, \qquad Z_u[t,:] = (z_t^u)^\top, \qquad \boldsymbol{\kappa} = (\kappa_1, \ldots, \kappa_C)^\top, \tag{38}$$

$$\widetilde{P}_<[t,i] = \begin{cases} \widetilde{q}_t^\top D_{t:i} k_i, & i < t, \\ 0, & i \geq t. \end{cases} \tag{39}$$

Here $\widetilde{Q}_\gamma \in \mathbb{R}^{C \times d_k}$, $Z_u \in \mathbb{R}^{C \times d_v}$, and $\widetilde{P}_< \in \mathbb{R}^{C \times C}$. Stacking the output rows gives

$$O = \widetilde{Q}_\gamma S_{\text{in}} + \widetilde{P}_< R^c + \text{Diag}(\boldsymbol{\kappa}) Z_u. \tag{40}$$

The outgoing state includes every commit in the block:

$$S_{\text{out}} = D_{C:0} S_{\text{in}} + \sum_{i=1}^{C} D_{C:i} k_i (\varepsilon_i^c)^\top. \tag{41}$$

Let $K_{\text{tail}} \in \mathbb{R}^{C \times d_k}$ have rows $K_{\text{tail}}[i,:] = (D_{C:i} k_i)^\top$. Then

$$S_{\text{out}} = \text{Diag}(\gamma_C) S_{\text{in}} + K_{\text{tail}}^\top R^c. \tag{42}$$

Both expressions follow directly from the tokenwise recurrence. Use controls the readout; the state passed to the next block contains commit updates.

Implementation conventions. We use 64-token computational blocks within training windows of at most 4096 tokens. State kernels compute commit residuals, and output kernels compute the strict-history and current-value terms. Backpropagation differentiates both terms with the same causal mask. All four quality configurations share this implementation, with the role gates tied for GDN2.

Queries and keys are L2-normalized before forming $\widetilde{q}_t$, which is not normalized again. The implementation multiplies both historical and current-value output terms once by $d_k^{-1/2}$, equivalent to absorbing this factor into q_t in the main text. Padding contributes no edit. Independent-window boundaries reset matrix states and short-convolution caches.

B Experimental Details

B.1 Model and Training

The table summarizes GDN2 and joint DisLA-GDN2. The write-only and erase-only configurations share the same backbone, SwiGLU width 1792, recurrent state size, rank 8, training budget, seed, and peak learning rate; each has 133,301,904 parameters.

Configuration	GDN2	DisLA
Layers / hidden width	12 / 768	12 / 768
Heads / key dimension / value dimension	12 / 64 / 64	12 / 64 / 64
SwiGLU width	1792	1792
Physical / active vocabulary	50,304 / 50,277	50,304 / 50,277
Tied embedding and output head	yes	yes
Disagreement rank	—	8
Total parameters	133,154,448	133,375,632
Matrix-state scalars per sequence	589,824	589,824
Short-convolution buffer length	4	4
Maximum training-window length	4096	4096
Training targets	3×10^9	3×10^9
Training seed / peak learning rate	123 / 0.0015	123 / 0.0015
Training GPU	RTX 4090 D	RTX 4090 D

Parameters, gradients, and AdamW states use FP32; training uses BF16 mixed precision with FP32 sensitive accumulations. AdamW has betas $(0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay 0.1, and global gradient clipping at 1. Weight decay excludes normalization parameters, biases, and designated decay/state parameters. Gradients are normalized by the total number of non-masked targets across accumulation steps. The final update uses its actual target count.

Ordinary linear, convolutional, and embedding weights use standard deviation 0.02, retaining architecture-specific initialization where defined. Residual-output projections are scaled by $1/\sqrt{2L}$, where $L = 12$ is the number of layers. The low-rank branch is initialized after the paired common parameters, so its addition does not change their initialization.

The parent–joint learning-rate trials use seed 17, 100M targets, and the prefix of the 3B-target schedule, including 30M-target warmup. Selection minimizes terminal validation NLL on 2,097,152 targets from 2,099 documents. Main runs begin from fresh initialization with seed 123 and complete 91,553 optimizer updates. Validation occurs at the first update reaching each 0.3B-target boundary and at completion. Checkpoint selection minimizes NLL, breaking ties toward the earlier checkpoint. The selected parent and joint checkpoints are terminal.

B.2 Language-Model Data

Training uses the FineWeb-Edu 100BT sample (Penedo et al., 2024) and the GPT-NeoX tokenizer (Black et al., 2022). Duplicate grouping uses Unicode NFKC and whitespace normalization; tokenization uses the original text. A salted content hash assigns 240 of 256 buckets to training, eight to validation, and eight to the held-out partition. Identical normalized texts share an identity. Training documents are screened against selected validation documents using word-five-gram MinHash retrieval followed by exact Jaccard similarity on retrieved candidates, rejecting matches at or above 0.8.

Long documents are divided into independent windows of at most 4096 input tokens. Tail windows retain valid next-token targets; prediction does not cross a document boundary. Both losses and training budgets count non-masked targets. The extra physical vocabulary rows are excluded from the predictive softmax. The independent confirmation set contains 8,388,608 targets from 8,070 documents and is not used for learning-rate or checkpoint selection.

For document d, let L_d be its summed loss and n_d its valid-target count. The reported aggregate is

$$\mathrm{NLL} = \frac{\sum_d L_d}{\sum_d n_d}. \tag{43}$$

All windows belonging to a sampled document stay together. Each paired bootstrap draw recomputes this ratio for both models, preserving target weighting across documents of different lengths.

B.3 Downstream Evaluation

Likelihood tasks. The LM Evaluation Harness (Gao et al., 2023) scores all ten tasks zero-shot, with maximum context 4096 and the training tokenizer. WikiText-2 uses rolling document likelihood and reports word perplexity. LAMBADA scores the final-word continuation. The remaining tasks rank answer choices; HellaSwag, ARC-Challenge, and OpenBookQA use length-normalized scores. Table 5 lists sources, splits, and primary metrics. No downstream score is used to select a learning rate or checkpoint.

RULER. We evaluate RULER (Hsieh et al., 2024) at the lengths and output budgets in Table 6, using 500 examples per task and length. Generation is greedy, in FP32, using the training chunk backend. We retain the task-native prompt templates, examples, and answer prefixes. Question-answering tasks accept any reference alias contained in the response; other tasks score the fraction of reference strings found in it. Scores are averaged over examples, then equally over the 13 tasks. The eight-task length curve always uses the same needle subset.

Real-world tasks. We use the Based/JRT adaptations of SWDE, FDA, and SQuAD 2.0 completion (Arora et al., 2024a;b; Rajpurkar et al., 2018), together with TriviaQA (Joshi et al., 2017), Natural Questions (Kwiatkowski et al., 2019), and DROP (Dua et al., 2019). SQuAD originated as

Table 5: Likelihood evaluation tasks. Acc. is answer accuracy; acc_n uses length-normalized answer likelihood. The nine-task mean includes LAMBADA and all eight reasoning tasks.

Task and source	Split	Primary metric
WikiText-2 (Merity et al., 2016)	test	word PPL
LAMBADA (Paperno et al., 2016)	test	PPL, acc.
PIQA (Bisk et al., 2020)	validation	acc.
HellaSwag (Zellers et al., 2019)	validation	acc_n
WinoGrande-XL (Sakaguchi et al., 2019)	validation	acc.
ARC-Easy (Clark et al., 2018)	test	acc.
ARC-Challenge (Clark et al., 2018)	test	acc_n
OpenBookQA (Mihaylov et al., 2018)	test	acc_n
Social IQa (Sap et al., 2019)	validation	acc.
BoolQ (Clark et al., 2019)	validation	acc.

Table 6: RULER evaluation protocol. Context and output budgets are in tokens. Each task–length condition contains 500 examples.

Family	Tasks	Context	Output
Needle retrieval	S-1/2/3, MK-1/2/3, MQ, MV	1K–8K	128
Variable tracking	VT	4K	30
Common-word extraction	CWE	4K	120
Frequent-word extraction	FWE	4K	50
Question answering	QA-1, QA-2	4K	32

Needle lengths are 1K, 2K, 4K, and 8K; S and MK denote single-needle and multi-key tasks.

an extractive question-answering task (Rajpurkar et al., 2016); Based's completion adaptation converts the question to a continuation prompt. Table 7 lists task forms and evaluation sizes. Inputs are capped at 2K tokens by retaining a context prefix together with the instruction and query. Generation uses a 48-token output budget and stops at EOS or the first newline; scoring uses the text before that newline. Greedy responses are scored by case-insensitive containment of a reference answer. For SQuAD, this is answer-containment accuracy rather than official exact match (EM); the other tasks also use this common scoring rule rather than their original leaderboard metrics.

Table 7: Real-world evaluation tasks. All tasks use a maximum 2K-token input and a 48-token output budget, with greedy decoding and reference-answer containment scoring.

Task	Task form	Examples
SWDE	Web attribute extraction	1,111
SQuAD completion	Question–answer completion	2,984
FDA	Document attribute extraction	1,102
TriviaQA	Question answering	1,688
Natural Questions	Question answering	3,157
DROP	Reading-comprehension QA	2,087

Paired statistics. Percentile intervals use 2000 resamples and seed 2027 for each task. Accuracy resamples matched examples; WikiText resamples its 62 documents and recomputes perplexity from total log probability and word count. Real-world tasks resample normalized-context clusters. Intervals are pointwise, without a multiple-comparison correction. The eight-task commonsense mean excludes LAMBADA; the main-table nine-task mean includes it. Neither mean includes perplexity.

C Complete Results and Analyses

Training trajectory. Figure 3 shows that the validation advantage persists across training. The difference is already negative at the first evaluation near 0.3B targets and ranges from roughly -0.003 to -0.008 nats across all ten observations.

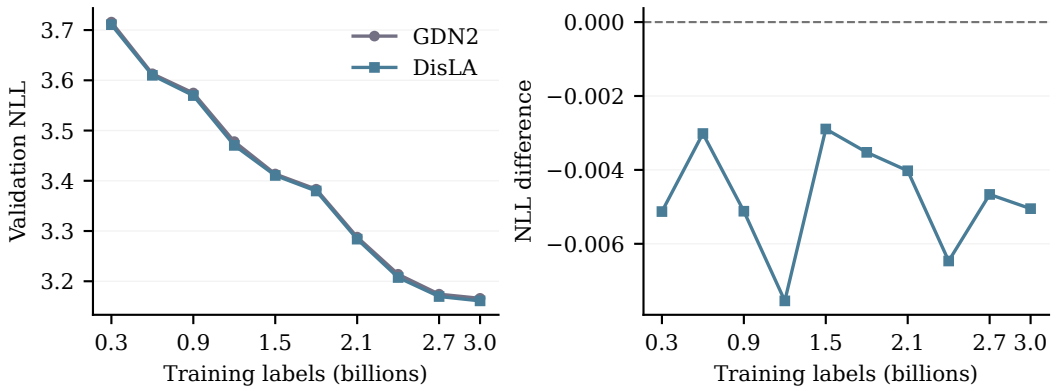


Figure 3: **Validation during pretraining.** Left: NLL at all ten scheduled evaluations. Right: DisLA minus GDN2 at matched training-target counts; negative values favor DisLA. Lines connect recorded evaluations without smoothing.

Language modeling and uncertainty. For the parent–joint pair, the selected and terminal checkpoints coincide. Their development-validation NLLs are 3.1660 and 3.1610, with paired difference interval $[-0.0063, -0.0038]$. On the independent confirmation set, NLL is 3.1596 for GDN2 and 3.1546 for DisLA, with interval $[-0.0057, -0.0044]$. The eight-task reasoning mean is 41.79% versus 43.21%. Table 8 reports pointwise intervals for WikiText perplexity and the nine accuracy metrics.

Table 8: **Pointwise paired uncertainty for language tasks.** Differences are DisLA minus GDN2. WikiText differences use word-perplexity units; accuracy differences use percentage points (pp). n counts documents for WikiText and examples otherwise.

Task	n	Metric	Δ	Paired 95% interval
WikiText-2	62	Word PPL	-1.313	$[-1.617, -1.026]$
LAMBADA	5,153	Acc. (pp)	-0.388	$[-1.223, +0.485]$
PIQA	1,838	Acc. (pp)	-0.435	$[-1.904, +1.034]$
HellaSwag	10,042	Norm. acc. (pp)	$+0.259$	$[-0.299, +0.807]$
WinoGrande	1,267	Acc. (pp)	$+2.131$	$[-1.105, +5.209]$
ARC-E	2,376	Acc. (pp)	$+0.084$	$[-1.641, +1.768]$
ARC-C	1,172	Norm. acc. (pp)	-0.853	$[-2.816, +1.109]$
OBQA	500	Norm. acc. (pp)	$+1.000$	$[-1.600, +3.800]$
SIQA	1,954	Acc. (pp)	-0.307	$[-1.844, +1.279]$
BoolQ	3,270	Acc. (pp)	$+9.511$	$[+7.217, +11.835]$

Complete in-context results. Table 2 reports all four configurations on every RULER task at 4K. Table 9 gives the four-configuration results on all eight needle tasks at every length, and Table 10 provides parent–joint paired task-level intervals for RULER and real-world evaluation.

Generation behavior. At 1K, S-NIAH-1 places a seven-digit passkey in repeated filler text. Of 500 saved responses, 468 from DisLA and 17 from GDN2 contain the filler sentence verbatim; 32 and 43, respectively, contain the correct passkey.

Table 9: **Needle tasks by context length.** Scores (%) on the same eight tasks, with 500 examples per condition. 8K is length extrapolation. Joint denotes DisLA-GDN2.

Task	Model	1K	2K	4K	8K
S-1	GDN2	**8.60**	**4.00**	0.80	0.60
	Joint	6.40	0.80	0.00	0.00
	Write-only	1.80	0.60	0.20	0.00
	Erase-only	3.60	2.00	**1.20**	**0.80**
S-2	GDN2	14.40	2.40	0.60	0.80
	Joint	19.20	6.00	3.80	1.40
	Write-only	12.20	**9.60**	**4.00**	**1.60**
	Erase-only	**20.00**	1.40	0.20	0.20
S-3	GDN2	0.00	0.00	0.00	0.00
	Joint	**4.40**	**0.20**	0.00	0.00
	Write-only	0.80	0.00	0.00	0.00
	Erase-only	0.60	**0.20**	0.00	**0.20**
MK-1	GDN2	**12.20**	9.80	3.20	2.40
	Joint	10.80	7.80	2.80	3.20
	Write-only	10.40	**11.20**	**4.20**	**4.00**
	Erase-only	9.80	7.20	2.60	2.40
MK-2	GDN2	0.00	**0.20**	0.00	0.00
	Joint	0.00	0.00	0.00	0.00
	Write-only	0.00	0.00	0.00	0.00
	Erase-only	0.00	0.00	0.00	0.00
MK-3	GDN2	0.00	0.00	0.00	0.00
	Joint	0.00	0.00	0.00	0.00
	Write-only	0.00	0.00	0.00	0.00
	Erase-only	0.00	0.00	0.00	0.00
MQ	GDN2	4.45	**5.20**	**0.50**	0.30
	Joint	**5.65**	4.15	0.05	**0.75**
	Write-only	3.00	2.30	0.20	0.10
	Erase-only	3.75	2.35	0.05	0.10
MV	GDN2	5.65	2.20	**0.45**	0.15
	Joint	6.95	**2.70**	0.05	**1.15**
	Write-only	**7.55**	0.55	0.00	0.10
	Erase-only	6.45	2.40	0.05	0.15

Table 10: **Pointwise paired uncertainty for in-context tasks.** Differences are DisLA minus GDN2 in percentage points; n counts examples. RULER resamples examples, and real-world tasks resample normalized-context clusters.

Suite	Task	n	Δ (pp)	Paired 95% interval (pp)
RULER@4K	S-1	500	-0.80	$[-1.60, -0.20]$
RULER@4K	S-2	500	$+3.20$	$[+1.60, +4.80]$
RULER@4K	S-3	500	$+0.00$	$[+0.00, +0.00]$
RULER@4K	MK-1	500	-0.40	$[-1.80, +1.00]$
RULER@4K	MK-2	500	$+0.00$	$[+0.00, +0.00]$
RULER@4K	MK-3	500	$+0.00$	$[+0.00, +0.00]$
RULER@4K	MQ	500	-0.45	$[-0.75, -0.15]$
RULER@4K	MV	500	-0.40	$[-0.70, -0.15]$
RULER@4K	VT	500	$+0.00$	$[+0.00, +0.00]$
RULER@4K	CWE	500	$+0.62$	$[+0.34, +0.92]$
RULER@4K	FWE	500	$+5.07$	$[+3.20, +7.00]$
RULER@4K	QA-1	500	$+1.60$	$[-0.20, +3.60]$
RULER@4K	QA-2	500	$+2.60$	$[+0.00, +5.00]$
Real@2K	SWDE	1111	-2.97	$[-4.32, -1.62]$
Real@2K	SQD	2984	$+1.07$	$[-0.30, +2.41]$
Real@2K	FDA	1102	$+0.09$	$[-0.64, +0.91]$
Real@2K	TQA	1688	$+1.13$	$[-0.83, +3.07]$
Real@2K	NQ	3157	$+0.00$	$[-0.76, +0.76]$
Real@2K	DROP	2087	$+1.39$	$[-0.15, +3.03]$

Learning-rate search. Table 11 reports terminal validation NLL for the six parent–joint trials. Both methods select 0.0015. Joint DisLA has lower NLL at the two smaller rates, whereas GDN2 has lower NLL at 0.003.

Table 11: Development learning-rate search: 100M targets per trial, seed 17. Lower NLL is better.

Learning rate	GDN2	DisLA	Difference
0.00075	4.043628	4.026866	-0.016762
0.00150	4.040978	4.024528	-0.016450
0.00300	4.062078	4.066188	$+0.004110$

D Resource Details

The joint DisLA-GDN2 rank-8 branch adds $8(768 + 1536) = 18{,}432$ weights per layer, or 221,184 across 12 layers. Its two projections are token-local. Persistent storage comprises the recurrent matrices and the three short-convolution buffers for queries, keys, and values. Each layer stores $12 \times 64 \times 64$ matrix scalars and $3 \times 768 \times 4$ convolution-buffer scalars. The length-four buffer is the allocated implementation layout. Thus all four configurations have 589,824 matrix scalars and 110,592 convolution-buffer scalars per sequence; byte counts depend on their respective dtypes.

Training also stores or recomputes activations and block-local intermediates, including the role logits and adjusted queries. These intermediates contribute to training memory but are not retained in the decoding cache.

Short-run training and prefill. The measurements in Table 4 use two isolated sessions per configuration on the same GPU. Each session initializes a fresh seed-17 model, performs two warmup and three timed training updates, and then measures batch-one 4K prefill with one warmup and three timed sequences. The driver version is 580.105.08. These probes use the training precision specified in Appendix B.1. Peak allocation refers to the training stage.

Selected-checkpoint decoding. Decoding uses the selected weights with BF16 autocast and the chunk-based single-token path, on driver 570.124.04. Each configuration has two sessions with a batch-one, 4096-token prompt. A session uses one warmup and three timed sequences, each containing three cached single-token steps on fixed continuation IDs, for 18 timed steps across both sessions. The probe times these cached model calls. The quality-generation evaluations use FP32 as specified in Appendix B.3.

Cache accounting. We count unique allocated storage for matrix and convolution states, avoiding double counting views. Under the measured layout, FP32 matrix states occupy $589{,}824 \times 4$ bytes and BF16 convolution states occupy $110{,}592 \times 2$ bytes, totaling 2,580,480 bytes, or 2520 KiB per sequence. Parameters, logits, temporary activations, and allocator reservations are excluded from this persistent-cache total. Training memory is reported separately.