SDG: STATE-DEPENDENT GATING AT CHUNK BOUNDARIES FOR LINEAR ATTENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent gated linear attention models use state-independent gates, computed from token representations without directly reading the recurrent memory. This design supports efficient chunkwise computation: sequences are divided into chunks of consecutive tokens, with parallel computation within each chunk. However, the same token representation produces the same gate, even when the layer stores different information. We introduce *State-Dependent Gating* (SDG), which computes element-wise gates from the change in memory over a chunk and applies them at the chunk boundary. This provides state feedback for forgetting while preserving parallel computation within each chunk. We develop fused kernels for Gated DeltaNet, Kimi Delta Attention, Gated DeltaNet-2, and Mamba-3. Our theoretical analysis quantifies the potential reconstruction benefits of state-dependent, element-wise gating under a signal–noise model. We also show that boundary decay is an element-wise ridge proximal step when the gate is held fixed. Empirical evaluation of 1.3B linear attention and hybrid models trained on 100B tokens shows improvements across language modeling, commonsense reasoning, synthetic and real-world retrieval, and long-context understanding. These results demonstrate that SDG is an effective approach to enhancing existing linear attention models.

1 INTRODUCTION

Linear attention stores context in a fixed-size recurrent state, enabling linear-time sequence modeling and constant-memory decoding (Katharopoulos et al., 2020; Schlag et al., 2021; Sun et al., 2023). Its limited capacity makes retention important: as new associations are written, they can interfere with those already stored (Kohonen, 1972; Arora et al., 2024a). Recent gated recurrent models control retention using coefficients predicted from token representations (Gu & Dao, 2023; Yang et al., 2023; 2025; Team et al., 2025; Hatamizadeh et al., 2026). These gates are data-dependent but state-independent: their coefficients do not directly read the recurrent memory of the same layer.

This separation supports efficient chunkwise computation. A sequence is divided into chunks of consecutive tokens, with parallel matrix operations within each chunk and recurrent state propagation between chunks. The gate coefficients can be computed before the state updates, and their structure enables further reuse of computation. Gated DeltaNet (GDN) uses one decay coefficient per head, whereas Kimi Delta Attention (KDA) uses one per key channel. Both share decay coefficients across value channels, allowing their chunkwise algorithms to reuse transition factors (Yang et al., 2025; Team et al., 2025).

The same design limits how gates use memory. For a fixed token representation, the gate is unchanged even if the layer stores different information. Although the delta update reads the state to correct stored associations, this read does not enter the gate predictor. Coefficient sharing also forces different state elements to use the same forgetting rate. Adding state-dependent, element-wise gates at every token, however, changes the structure used by existing chunkwise algorithms. Direct state feedback makes each gate depend on preceding state updates, while generic element-wise decay removes the transition sharing used by these kernels.

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.

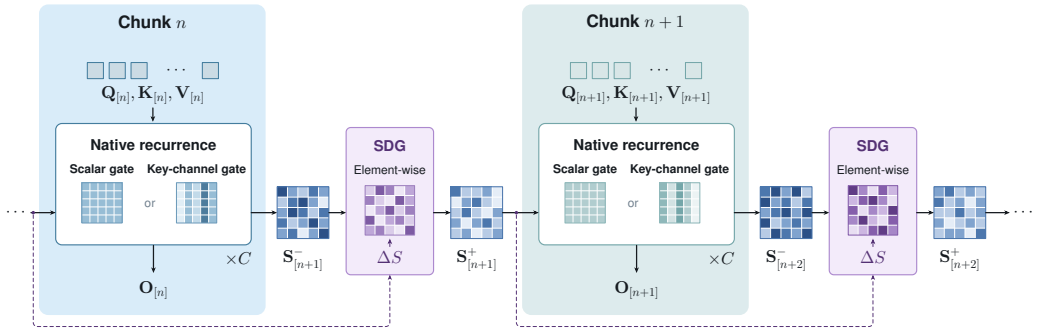


Figure 1: **State-Dependent Gating (SDG).** The base layer processes a chunk of C tokens, taking the incoming state $\mathbf{S}^+_{[n]}$ to the updated state $\mathbf{S}^-_{[n+1]}$ and producing outputs $\mathbf{O}_{[n]}$. SDG uses the difference between states to compute an element-wise gate and passes the new state $\mathbf{S}^+_{[n+1]}$ to the next chunk.

These constraints motivate our central question:

> **Question.** *How can we introduce state-dependent, element-wise forgetting while preserving parallel computation within each chunk?*

We address this question with *State-Dependent Gating* (SDG), which applies the additional gate only at chunk boundaries (Figure 1). After the base layer processes a chunk, SDG subtracts the incoming state from the updated state and uses this difference to compute a separate retention coefficient for each state element. For these gated recurrences, the difference reflects how the chunk modifies the incoming memory through writing, erasing, and forgetting. The gate then scales the updated state before it enters the next chunk. Because the current chunk's outputs have already been computed, its algorithm and parallel factorization remain unchanged. Both boundary states are available from the recurrence; the added computation evaluates and applies the boundary gate.

We support this design with theoretical analysis and empirical evaluation. Under a signal–noise model, our analysis compares fixed and shared gates with an ideal gate that can vary across boundary observations and state elements. This quantifies the potential reconstruction benefits of adaptive, element-wise decay. We assess the learned method across recurrent and hybrid architectures, including controls that replace state-change feedback with chunk information alone.

Our contributions are summarized as follows:

- **Boundary gating and fused implementation.** We introduce element-wise decay from chunk-level state changes, applied at boundaries to preserve the base layer's within-chunk algorithm. We develop fused kernels for GDN, KDA, Gated DeltaNet-2, and Mamba-3.
- **Theoretical characterization.** Under a signal–noise model, we derive an exact decomposition of the excess reconstruction error of fixed and shared boundary gates. We also characterize boundary decay, with the realized gate held fixed, as an element-wise ridge proximal step.
- **Empirical evaluation.** At the 1.3B scale with 100B training tokens, SDG improves WikiText perplexity and average real-world retrieval across four recurrent backbones. Retrieval gains extend to three sliding-window hybrids, while reasoning and broader long-context results are mixed. Ablations examine gate inputs, granularity, and projection, and runtime measurements quantify the cost of boundary gating.

A comprehensive review of **Related work** is provided in Appendix A.

2 Preliminaries

2.1 Recurrent state and native gates

For one head, queries and keys $\boldsymbol{q}_t, \boldsymbol{k}_t \in \mathbb{R}^{d_k}$ read and write a state $\mathbf{S}_t \in \mathbb{R}^{d_v \times d_k}$, with value channels as rows and key channels as columns. Values and outputs satisfy $\boldsymbol{v}_t, \boldsymbol{o}_t \in \mathbb{R}^{d_v}$. The unnormalized

additive recurrence is (Katharopoulos et al., 2020; Schlag et al., 2021)

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^\top, \qquad \boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t. \tag{1}$$

The read includes the current write. Sequences start from $\mathbf{S}_0 = \mathbf{0}$. The state size is independent of sequence length.

In GDN, KDA, and GDN-2, native gates depend on token representations, whereas delta corrections read the recurrent state. GDN uses scalar decay, KDA uses key-channel decay, and GDN-2 separates erase and write gates. For fixed inputs, these layers share a key-space transition across value rows. Their token-dependent factors can therefore be prepared before the state scan. Appendix F.1 gives their token updates. Mamba-3 instead uses a $d_v \times d_s$ state without delta correction (Appendix F.8).

2.2 Chunkwise computation

Training and prefill scan states across chunks with parallel computation within each chunk. Decoding uses the native token recurrence. For the formulas below, assume $L = NC$ with N complete chunks of C tokens. The incoming state is $\mathbf{S}_{[n]} = \mathbf{S}_{(n-1)C}$. Additive and gated delta-rule chunks have a shared transition $\mathbf{P}_{[n]} \in \mathbb{R}^{d_k \times d_k}$ and a zero-memory response $\mathbf{\Omega}_{[n]} \in \mathbb{R}^{d_v \times d_k}$:

$$\begin{aligned} \mathbf{S}_{[n+1]} &= \mathbf{S}_{[n]} \mathbf{P}_{[n]} + \mathbf{\Omega}_{[n]}, \\ \mathbf{S}_{[n+1]} - \mathbf{S}_{[n]} &= \mathbf{S}_{[n]} (\mathbf{P}_{[n]} - \mathbf{I}) + \mathbf{\Omega}_{[n]}. \end{aligned} \tag{2}$$

For fixed token inputs and native gates, both factors are independent of incoming memory. Thus gated state changes include the incoming-memory response; additive updates have $\mathbf{P}_{[n]} = \mathbf{I}$. Appendix F.1 expands the additive computation, and Appendix F.3 derives the gated forms.

3 State-Dependent Gating

3.1 Element-wise decay at chunk boundaries

SDG applies element-wise decay after the native chunk recurrence $\mathcal{F}$, retaining its token gates and within-chunk computation. Let $\mathbf{S}^+_{[n]}$ be the incoming state, $\mathbf{S}^-_{[n+1]}$ the updated state, and $\mathbf{S}^+_{[n+1]}$ the gated state. $\mathcal{X}_{[n]}$ contains projected token inputs and native gates, and $\boldsymbol{\xi}_{[n]}$ denotes auxiliary state. The computation returns the chunk outputs $\mathbf{O}_{[n]} \in \mathbb{R}^{C \times d_v}$:

$$\begin{aligned} (\mathbf{O}_{[n]}, \mathbf{S}^-_{[n+1]}, \boldsymbol{\xi}_{[n+1]}) &= \mathcal{F}(\mathcal{X}_{[n]}, \mathbf{S}^+_{[n]}, \boldsymbol{\xi}_{[n]}), \\ \mathbf{\Lambda}_{[n]} &= \mathcal{G}_\theta(\mathbf{S}^+_{[n]}, \mathbf{S}^-_{[n+1]}), \qquad \mathbf{\Lambda}_{[n]} \in (0,1]^{d_v \times d_k}, \\ \mathbf{S}^+_{[n+1]} &= \mathbf{\Lambda}_{[n]} \odot \mathbf{S}^-_{[n+1]}. \end{aligned} \tag{3}$$

Decay scales the entire updated state, including the current chunk's writes. Outputs are formed first, so the gate affects only subsequent chunks. Native auxiliary states advance unchanged.

Token-level state feedback prevents gate precomputation from token representations alone, while general element-wise decay removes the shared value-row transitions of Section 2. Boundary placement preserves native within-chunk factorization and confines the additional dependency to the inter-chunk scan (Appendix H).

3.2 Gate parameterization

The gate uses the state change, which includes both the incoming-memory response and the chunk contribution: $\Delta\mathbf{S}_{[n]} = \mathbf{S}^+_{[n]}(\mathbf{P}_{[n]} - \mathbf{I}) + \mathbf{\Omega}_{[n]}$ by Eq. (2). Each head learns an unconstrained projection $\mathbf{W}_S \in \mathbb{R}^{d_k \times d_k}$ across key channels within each value row and an unconstrained element-wise gain $\mathbf{G} \in \mathbb{R}^{d_v \times d_k}$. These predict the gate without directly projecting the stored state:

$$\Delta\mathbf{S}_{[n]} = \mathbf{S}^-_{[n+1]} - \mathbf{S}^+_{[n]}, \quad \mathbf{X}_{[n]} = \Delta\mathbf{S}_{[n]} \mathbf{W}_S^\top, \quad \mathbf{Z}_{[n]} = \mathbf{G} \odot \mathbf{X}_{[n]}, \tag{4}$$

$$\boldsymbol{\rho}_{[n]} = \mathbf{Z}_{[n]} + \sqrt{\mathbf{Z}_{[n]} \odot \mathbf{Z}_{[n]} + \rho_0^2 \mathbf{1}}, \tag{5}$$

$$\mathbf{\Lambda}_{[n]} = \exp(-\boldsymbol{\rho}_{[n]}), \qquad \mathbf{S}^+_{[n+1]} = \mathbf{\Lambda}_{[n]} \odot \mathbf{S}^-_{[n+1]}. \tag{6}$$

Equation (5) is twice Squareplus (Barron, 2021), applied element-wise (definition and derivatives in Appendix I). Positive scores increase decay, while negative scores reduce it. For finite scores, $0 < \mathbf{\Lambda}_{[n]} < 1$. We initialize $\mathbf{G} = \mathbf{0}$ and fix $\rho_0 = -\log \lambda_0$, $\lambda_0 = 0.9999$, so all gates initially equal $\lambda_0 \mathbf{1}$. Appendix C gives the remaining initialization details.

Algorithm 1 shows the computation order, with the added gate steps in bold. The native recurrence $\mathcal{F}$ retains each backbone's state and output computations. The algorithm assumes complete chunks, $L = NC$. For arbitrary lengths and cached continuations, Appendix F.9 specifies the scan. Incomplete final chunks do not trigger boundary decay.

Algorithm 1 SDG: Boundary-Gated Chunk Scan

Require: Sequence length $L = NC$, and $N = L/C$ complete chunks, native token quantities $\mathcal{X}$, initial state $\mathbf{S}^+_{[1]}$ and auxiliary state $\boldsymbol{\xi}_{[1]}$, chunk size C, gate parameters $\mathbf{W}_S$, $\mathbf{G}$, and fixed base rate ρ_0.

1: Partition $\mathcal{X}$ into N complete chunks of C tokens, $\{\mathcal{X}_{[n]}\}_{n=1}^{N}$.
2: **for** $n = 1, \ldots, N$ **do**
3: **Retain the incoming state** $\mathbf{S}^{\mathrm{src}} \leftarrow \mathbf{S}^+_{[n]}$**.**
4: Compute $(\mathbf{O}_{[n]}, \mathbf{S}^-_{[n+1]}, \boldsymbol{\xi}_{[n+1]}) \leftarrow \mathcal{F}(\mathcal{X}_{[n]}, \mathbf{S}^+_{[n]}, \boldsymbol{\xi}_{[n]})$.
5: **Compute** $\Delta\mathbf{S}_{[n]} \leftarrow \mathbf{S}^-_{[n+1]} - \mathbf{S}^{\mathrm{src}}$**.**
6: **Compute** $\mathbf{X}_{[n]} \leftarrow \Delta\mathbf{S}_{[n]}\mathbf{W}_S^\top$**,** $\mathbf{Z}_{[n]} \leftarrow \mathbf{G} \odot \mathbf{X}_{[n]}$**.**
7: **Compute** $\boldsymbol{\rho}_{[n]} \leftarrow 2\,\mathrm{squareplus}(\mathbf{Z}_{[n]}; \rho_0^2)$**,** $\mathbf{\Lambda}_{[n]} \leftarrow \exp(-\boldsymbol{\rho}_{[n]})$**.**
8: **Set** $\mathbf{S}^+_{[n+1]} \leftarrow \mathbf{\Lambda}_{[n]} \odot \mathbf{S}^-_{[n+1]}$**.**
9: **end for**
10: **return** $\mathrm{Concat}(\mathbf{O}_{[1]}, \ldots, \mathbf{O}_{[N]})$, $\mathbf{S}^+_{[N+1]}$, and $\boldsymbol{\xi}_{[N+1]}$.

3.3 Fused kernel implementation

The boundary update is fused into the inter-chunk state scan using Triton (Tillet et al., 2019), while retaining native factor preparation and output kernels. Appendix I gives forward and backward kernel details, with reference algorithms for additive linear attention in Algorithms 2 and 3, respectively. Each head adds $O(d_v d_k^2)$ projection and $O(d_v d_k)$ element-wise work per boundary, amortized over C tokens, plus backward computation (Appendix J). For Mamba-3, replace d_k by d_s. Cached decoding retains an additional source matrix and token counter. We use $C = 64$ to balance recurrent overhead and on-chip memory usage, matching the chunk size of the adopted Flash Linear Attention kernels (Yang & Zhang, 2024). We apply the same boundary rules during training, prefill, and decoding.

4 Analysis

We characterize the realized boundary operation, then quantify potential gains from ideal reconstruction gates under a fixed upstream state distribution.

4.1 Boundary decay as an adaptive element-wise ridge

Holding the realized gate fixed, boundary decay is an element-wise ridge proximal step.

Remark 1 (Adaptive element-wise ridge)**.** Fix $\boldsymbol{\rho}_{[n]} \in \mathbb{R}^{d_v \times d_k}_{>0}$, let $\mathbf{\Lambda}_{[n]} = \exp(-\boldsymbol{\rho}_{[n]})$, and set $\boldsymbol{\kappa} = \exp(\boldsymbol{\rho}_{[n]}) - \mathbf{1}$ element-wise. Then for any state $\mathbf{S}^-_{[n+1]}$ before boundary decay,

$$\mathbf{S}^+_{[n+1]} = \mathbf{\Lambda}_{[n]} \odot \mathbf{S}^-_{[n+1]} = \underset{\mathbf{M} \in \mathbb{R}^{d_v \times d_k}}{\arg\min} \left\{ \tfrac{1}{2} \left\| \mathbf{M} - \mathbf{S}^-_{[n+1]} \right\|_F^2 + \tfrac{1}{2} \sum_{a,b} \kappa_{ab} M_{ab}^2 \right\}. \tag{7}$$

The rates set separate ridge weights, with $\boldsymbol{\kappa} = \boldsymbol{\rho}_{[n]} + O(\boldsymbol{\rho}_{[n]}^{\odot 2})$ at small rates. This identity applies to all four backbones and concerns only the boundary operation. Composing it with a chunk's token updates is not generally one forward–backward step. Appendix G.1 gives the proof, converse, and backbone-specific token-update interpretations.

4.2 OPTIMAL DECAY AND THE COST OF SHARING IT

At one boundary, keep the upstream distribution fixed and model the state as useful memory plus residual interference (Kohonen, 1972; Schlag et al., 2021):

$$\mathbf{S}^- = \mathbf{S}^\star + \mathbf{N}, \tag{8}$$

Here $\mathbf{S}^\star$ is useful to future queries and $\mathbf{N}$ is residual interference; neither is separately observed by the learned gate. For oracle information Φ, a gate is *admissible* if it is Φ-measurable with finite reconstruction risk $\mathcal{R}(\mathbf{\Lambda}) = \mathbb{E}\|\mathbf{\Lambda} \odot \mathbf{S}^- - \mathbf{S}^\star\|_F^2$. An independent isotropic query makes this risk proportional to readout error; other query models can change the optimum (Appendix G.3).

Lemma 1 (Conditional shrinkage and excess risk). *Assume that $\mathbf{S}^\star$ and $\mathbf{N}$ have finite second moments and that $\mathbf{m}_- = \mathbb{E}[\mathbf{S}^- \odot \mathbf{S}^- \mid \Phi] > \mathbf{0}$ almost surely, and define*

$$\mathbf{\Lambda}^\star = \mathbb{E}[\mathbf{S}^\star \odot \mathbf{S}^- \mid \Phi] \oslash \mathbf{m}_-. \tag{9}$$

Then every admissible gate satisfies

$$\mathcal{R}(\mathbf{\Lambda}) = \mathcal{R}(\mathbf{\Lambda}^\star) + \mathbb{E}\big\|\mathbf{m}_-^{1/2} \odot (\mathbf{\Lambda} - \mathbf{\Lambda}^\star)\big\|_F^2. \tag{10}$$

If in addition $\mathbb{E}[\mathbf{S}^\star \odot \mathbf{N} \mid \Phi] = \mathbf{0}$, then $\mathbf{\Lambda}^\star = \mathbf{m}_\star \oslash (\mathbf{m}_\star + \mathbf{m}_N) \in [0,1]^{d_v \times d_k}$, with $\mathbf{m}_\star = \mathbb{E}[\mathbf{S}^\star \odot \mathbf{S}^\star \mid \Phi]$, $\mathbf{m}_N = \mathbb{E}[\mathbf{N} \odot \mathbf{N} \mid \Phi]$ and $\mathbf{m}_- = \mathbf{m}_\star + \mathbf{m}_N$. All operations are element-wise.

We compare fixed boundary coefficients with the adaptive oracle under the same state distribution. The fixed scalar and per-key families below describe coefficient sharing, not native GDN or KDA gates, which can vary with token inputs.

Corollary 1 (Cost of fixing and of sharing the gate). *Let $\bar{\mathbf{m}} = \mathbb{E}[\mathbf{m}_-]$ be finite and strictly positive. Consider*

$$\mathcal{C}_{\text{scalar}} = \{\lambda \mathbf{1}\} \subset \mathcal{C}_{\text{key}} = \{\mathbf{1}_{d_v} \boldsymbol{\lambda}^\top\} \subset \mathcal{C}_{\text{elem}} = \mathbb{R}^{d_v \times d_k} \subset \mathcal{C}_{\text{adapt}} = \{\textit{admissible gates}\},$$

where the first three contain only constants. Define the optimal constant element-wise gate and its weighted averages by

$$\bar{\mathbf{\Lambda}} = \mathbb{E}[\mathbf{S}^\star \odot \mathbf{S}^-] \oslash \bar{\mathbf{m}}, \quad \bar{m}_{\cdot b} = \sum_a \bar{m}_{ab},$$

$$\mu_b = \frac{\sum_a \bar{m}_{ab} \bar{\Lambda}_{ab}}{\bar{m}_{\cdot b}}, \qquad \mu = \frac{\sum_b \bar{m}_{\cdot b} \mu_b}{\sum_b \bar{m}_{\cdot b}}.$$

Then $\bar{\mathbf{\Lambda}}$, $\mathbf{1}_{d_v} \boldsymbol{\mu}^\top$, and $\mu \mathbf{1}$ minimize $\mathcal{R}$ over the constant element-wise, per-key, and scalar families, respectively, and

$$\min_{\mathcal{C}_{\text{scalar}}} \mathcal{R} - \min_{\mathcal{C}_{\text{adapt}}} \mathcal{R} = \underbrace{\mathbb{E}\big\|\mathbf{m}_-^{1/2} \odot (\mathbf{\Lambda}^\star - \bar{\mathbf{\Lambda}})\big\|_F^2}_{\mathcal{E}_{\text{state}}} + \underbrace{\sum_{a,b} \bar{m}_{ab} (\bar{\Lambda}_{ab} - \mu_b)^2}_{\mathcal{E}_{\text{value}}} + \underbrace{\sum_b \bar{m}_{\cdot b} (\mu_b - \mu)^2}_{\mathcal{E}_{\text{key}}}. \tag{11}$$

Dropping $\mathcal{E}_{\text{key}}$ gives the per-key family's excess risk; dropping the last two gives that of the constant element-wise family. If $\mathbf{\Lambda}^\star \in [0,1]^{d_v \times d_k}$ almost surely, all four minimizers lie in this box. Constraining the families to the box then leaves Eq. (11) unchanged.

The three nonnegative costs respectively arise from fixing coefficients across observations, sharing those fixed coefficients across value rows, and further sharing across key channels. Each vanishes when the corresponding optimal coefficients already satisfy the restriction. Appendix G.2 gives the conditional-shrinkage and projection proofs, oracle-rate endpoints, and adaptive-sharing counterpart. Appendix G.3 illustrates when costs are positive.

These are potential gains from adapting to Φ. The first term alone does not measure information beyond token inputs. Constant scalar/per-key families are not native token-dependent GDN/KDA gates. The implemented gate uses only $\Delta \mathbf{S}_{[n]}$ through a restricted parameterization, which need not contain the full boundary-state pair's information. The analysis neither identifies the gate found by training nor predicts trained-model ordering. Moreover, the risk in Eq. (11) is defined on unnormalized states. Because a non-uniform gate can also change a readout's direction, the decomposition does not by itself characterize prediction error after RMS normalization. Table 4 evaluates the gate choices empirically; Appendix G provides the complete proofs and additional results.

5 EXPERIMENTS

We evaluate SDG's effects on prediction quality, the contributions of its gate design, and its computational overhead. We also examine the local output changes caused by a single boundary update.

5.1 EXPERIMENTAL SETUP

We train paired GDN, KDA, Mamba-3 (SISO), and GDN-2 models on 100B FineWeb-Edu tokens (Lozhkov et al., 2024), using sequences of 4096 tokens and chunk size $C = 64$. Each model has approximately 1.3B non-embedding parameters. Its +SDG counterpart adds a boundary gate to every recurrent layer. Table 6 gives parameter counts. Each pair shares the data order, optimizer settings, and initial values of shared parameters. Training details are in Appendix D.

Evaluation covers language modeling, nine zero-shot accuracy tasks, four RULER needle-in-a-haystack tasks, six real-world retrieval tasks, and all 21 LongBench tasks. We also evaluate hybrid GDN, KDA, and GDN-2 models that alternate recurrent layers with 2K sliding-window attention.

5.2 PERFORMANCE ACROSS BACKBONES

Language modeling and zero-shot accuracy. SDG lowers WikiText perplexity by 0.10–0.58 across the four recurrent backbones (Table 1). LAMBADA perplexity improves for three models and rises slightly for KDA. Average zero-shot accuracy increases for GDN, Mamba-3, and GDN-2, while KDA is unchanged at the reported precision. Task-level effects remain mixed, including lower BoolQ accuracy in all four pairs.

Table 1: Language modeling and zero-shot accuracy. Avg. is the mean over the nine accuracy columns; acc_n denotes length-normalized accuracy. Bold marks the better result within each pair.

Model	Wiki. ppl ↓	LMB. ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	OBQA acc ↑	SIQA acc ↑	BoolQ acc ↑	Avg. acc ↑
GDN	16.57	11.87	46.28	72.20	44.03	**56.67**	70.62	**35.84**	30.80	40.63	**59.54**	50.73
GDN+SDG	**15.99**	**11.64**	**47.70**	**73.67**	**44.23**	56.43	**72.39**	35.67	**31.20**	**40.69**	58.78	**51.20**
KDA	16.01	**11.11**	48.57	**72.52**	43.50	**57.37**	70.37	35.84	**31.40**	40.63	**60.31**	**51.17**
KDA+SDG	**15.91**	11.15	**49.04**	72.47	**43.91**	56.60	**70.50**	**36.43**	**31.40**	**40.80**	59.39	**51.17**
Mamba-3	15.55	11.93	46.98	72.91	44.83	55.09	72.80	36.52	**31.20**	40.58	**60.55**	51.27
Mamba-3+SDG	**15.34**	**11.37**	**48.09**	**73.18**	**44.93**	**57.30**	**72.98**	**37.37**	30.80	**40.94**	59.91	**51.72**
GDN-2	15.57	11.55	47.72	**73.23**	44.31	58.17	71.38	36.01	**30.40**	40.58	**60.09**	51.32
GDN-2+SDG	**15.40**	**11.31**	**47.89**	73.07	**44.80**	**59.51**	**72.47**	**37.20**	**30.40**	**40.89**	58.53	**51.64**

In-context retrieval on synthetic data. On the evaluated RULER (Hsieh et al., 2024) tasks, multi-key retrieval improves across all four backbones at 2K and 4K (Table 2). At 4K, MK-NIAH-1 gains are 0.2, 4.4, 13.6, and 9.8 percentage points for GDN, KDA, Mamba-3, and GDN-2, respectively. Single-needle results are less uniform: KDA falls from 36.4% to 24.2% on S-NIAH-3 at 4K. Thus, improvements vary even within the task length. The two tasks evaluated at 8K also give mixed results, so these gains do not establish better length extrapolation.

In-context retrieval on real-world tasks. Following Arora et al. (2024b), we evaluate six real-world retrieval tasks (Table 3). SDG raises reported mean accuracy by 0.39–0.83 percentage points across the four backbones.

Long-context understanding. On LongBench (Bai et al., 2024), SDG improves average performance for GDN, KDA, and Mamba-3. Notably, code accuracy shows consistent great gains across all four models. Appendix E.1 reports the full results and analysis.

Hybrid models. Figure 2 shows results for hybrid GDN, KDA, and GDN-2. SDG lowers LAMBADA perplexity and improves average real-world retrieval accuracy across all three models. S-NIAH-3 accuracy also improves at 4K. LongBench scores increases for KDA and GDN-2. These results suggest that SDG can complement hybrid architectures. Appendix E.2 reports the full results.

Table 2: RULER retrieval accuracy (%) for recurrent models trained at 4K. Columns give input lengths. Bold marks the better result within each baseline–SDG pair.

Model	S-NIAH-1				S-NIAH-2				S-NIAH-3			MK-NIAH-1		
	1K	2K	4K	8K	1K	2K	4K	8K	1K	2K	4K	1K	2K	4K
GDN	99.8	**100**	**100**	**84.6**	99.8	**100**	85.2	34.0	82.8	63.4	39.6	39.8	31.6	24.0
GDN+SDG	**100**	**100**	**100**	78.6	**100**	**100**	**93.0**	**39.2**	**90.4**	**70.6**	**49.8**	**40.8**	**33.6**	**24.2**
KDA	**100**	**100**	94.0	83.6	**100**	**100**	89.4	32.0	83.4	68.2	**36.4**	37.8	31.6	22.4
KDA+SDG	**100**	**100**	**98.6**	**87.2**	**100**	99.4	**92.6**	**37.8**	**91.4**	**72.8**	24.2	**38.8**	**34.0**	**26.8**
Mamba-3	99.8	99.4	86.0	**46.6**	99.6	99.6	76.6	**34.4**	66.4	54.4	22.2	**59.4**	52.6	26.0
Mamba-3+SDG	**100**	**100**	**92.8**	39.6	**100**	**99.8**	**78.8**	**34.4**	**75.2**	**66.2**	**25.6**	58.0	**54.8**	**39.6**
GDN-2	**100**	**100**	**100**	**99.8**	**100**	99.6	**93.2**	35.2	**94.6**	84.2	37.8	45.4	43.8	30.4
GDN-2+SDG	**100**	**100**	**100**	94.8	**100**	**99.8**	90.0	**53.4**	**94.6**	**92.0**	**49.2**	**61.0**	**63.4**	**40.2**

Table 3: Real-world retrieval accuracy (%) for recurrent models. Avg. is the mean over the six tasks. SQD, TQA, and NQ denote SQuAD, TriviaQA, and Natural Questions. Best results within each model family are bold (ties included).

Model	SWDE	SQD	FDA	TQA	NQ	DROP	Avg.
GDN	24.37	**37.92**	**23.07**	60.19	21.16	20.99	31.28
GDN+SDG	**26.80**	37.15	22.42	**60.84**	**21.82**	**22.40**	**31.91**
KDA	26.34	38.26	23.62	**62.68**	**23.12**	**21.18**	32.53
KDA+SDG	**27.74**	**38.97**	**24.89**	62.62	22.71	20.60	**32.92**
Mamba-3	27.93	**37.05**	**26.25**	61.73	21.03	21.47	32.58
Mamba-3+SDG	**29.90**	36.98	25.79	**62.38**	**22.17**	**21.75**	**33.16**
GDN-2	27.93	37.76	**25.52**	61.85	21.95	**21.80**	32.80
GDN-2+SDG	**31.02**	**37.79**	25.40	**63.09**	**22.65**	**21.80**	**33.63**

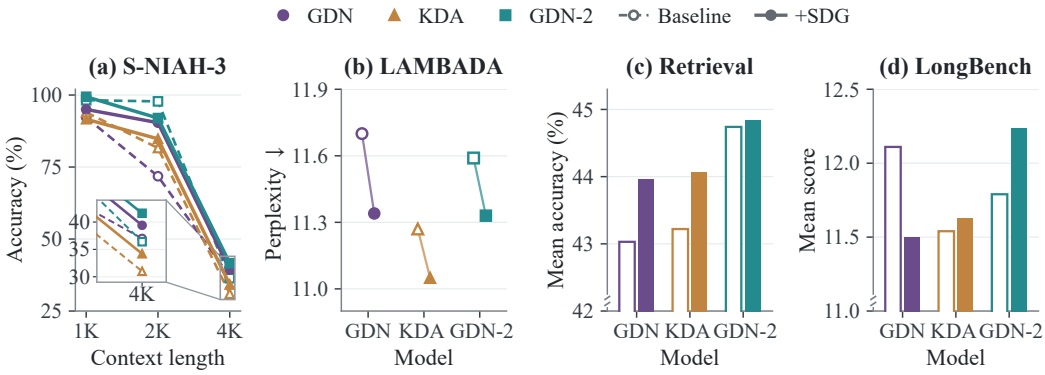


Figure 2: S-NIAH-3 accuracy, LAMBADA perplexity, mean six-task retrieval accuracy, and mean 21-task LongBench score on hybrid models. Open and filled marks denote baseline and +SDG.

5.3 Ablations and Boundary Diagnostics

Gate inputs. Table 4 compares gate inputs, coefficient sharing, and projection on GDN. Fixed boundary decay and a learned static gate score below SDG on all four metrics. To examine within-layer state feedback, the chunk-input control replaces $\Delta\mathbf{S}_{[n]}$ with $\mathbf{\Omega}_{[n]}$, the native chunk contribution from a zero incoming state. For fixed chunk inputs, $\mathbf{\Omega}_{[n]}$ excludes the response of that layer's incoming memory, while token representations can still carry context from earlier layers. The dynamic inputs use the same gate parameterization. Constructing $\mathbf{\Omega}_{[n]}$ requires an additional matrix product, whereas $\Delta\mathbf{S}_{[n]}$ uses a subtraction of two available states. Appendix E.3 compares their GDN operator latencies in forward and backward computation.

Table 4: Ablations of SDG on GDN. NIAH@4K averages S-NIAH-2, S-NIAH-3, and MK-NIAH-1 at 4K. Bold marks the highest score in each metric.

Variant	Commonsense avg ↑	NIAH @4K ↑	Recall avg ↑	LongBench avg ↑
Gate granularity				
GDN	50.73	49.60	31.28	10.37
GDN+SDG (Channel-wise)	51.06	54.20	31.69	10.71
GDN+SDG (Element-wise)	**51.20**	**55.67**	**31.91**	10.95
Gate input				
Fixed boundary decay	50.89	42.87	28.87	9.79
Learned static gate	50.76	48.73	29.32	10.03
Gate input $\mathbf{\Omega}$	51.15	55.00	31.89	**11.13**
Gate input $\mathbf{S}^{+}_{[n]}$	50.94	52.60	30.84	10.91
Gate projection				
Fixed $\mathbf{W}_S = \mathbf{I}$	50.99	51.13	31.82	10.86

The $\mathbf{\Omega}$ control recovers most of SDG's gains. Using $\Delta\mathbf{S}_{[n]}$ instead raises NIAH@4K by 0.67 points and real-world retrieval by 0.02 points, but lowers LongBench by 0.18 points. The difference between these two inputs is modest and task-dependent. Using the incoming state $\mathbf{S}^{+}_{[n]}$ gives lower scores than the state-change input on all four metrics. The reported scores are point estimates without uncertainty estimates; small gaps do not establish a statistically reliable advantage.

Sharing and projection. Element-wise gating scores above channel-wise gating on all four metrics, including a 1.47-point gain on NIAH@4K. The channel-wise variant shares one retention coefficient per key channel across all value rows and retains the same parameter count as element-wise gating. Learning $\mathbf{W}_S$ also improves all four scores relative to fixing $\mathbf{W}_S = \mathbf{I}$, with a 4.54-point gain on NIAH@4K. The static and identity-projection variants use fewer parameters. Appendix E.3 details these controls and costs.

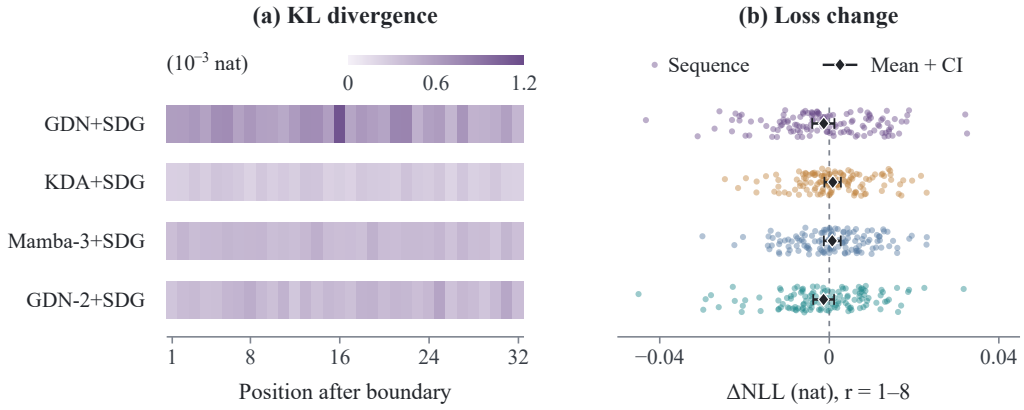


Figure 3: Effects of one SDG boundary update, with 128 observations per model. **(a)** The heatmap shows mean KL divergence from skip to normal at each of the first 32 affected positions. **(b)** Colored points show sequence means of ΔNLL over positions 1–8, defined as normal minus skip. Black diamonds and bars show means and bootstrap intervals adjusted across four models for approximately 95% simultaneous coverage.

Boundary output stability. We compare normal execution with skipping one boundary update across all recurrent layers of the same trained model. The two branches start from identical caches and consume the same subsequent ground-truth tokens. Over the first eight affected positions, top-1 predictions agree at 98.34–99.12%, and mean Kullback–Leibler (KL) divergence is below 6.5×10^{-4}. The four-model-adjusted intervals for mean negative log-likelihood (NLL) change all contain zero, with upper bounds below 0.003 nat per token (Figure 3).

These small averages coexist with individual loss increases and decreases. They do not bound per-token changes or characterize free-running generation. Mamba-3 contributes summary statistics. Sample matching and intervention controls were checked for GDN, KDA, and GDN-2. We also use valid filler tokens to place answer generation before and after chunk boundaries across benchmarks, with absolute changes in accuracy of at most 0.5 percentage points on commonsense reasoning and 1 percentage point on retrieval. Appendix E.5 gives the protocol and results.

5.4 Efficiency

Training throughput. We measure end-to-end training throughput for recurrent-only models on a single NVIDIA H200, with 16,384 input tokens per optimizer step. Across the tested 2K–16K sequence lengths, SDG retains 85.0–89.4% of baseline throughput for GDN, KDA, and GDN-2, and 72.5–75.1% for Mamba-3. Preserving within-chunk computation therefore still incurs a measurable training cost. GDN and GDN-2 with SDG show a smaller throughput decline across the tested lengths than the full-attention reference (Figure 4). See Appendix D.4 for details.

Inference latency. Inference measurements use hybrids with alternating recurrent and 2K SWA layers, on a single NVIDIA H200 at batch size one. For GDN and GDN-2 with SDG, time to first token (TTFT) grows approximately linearly at long contexts. Time per output token (TPOT) stays nearly constant for prompts from 512 to 512K tokens (Figure 4). At 256K, SDG adds 0.23 and 0.25 seconds to their TTFT, yielding 1.75 and 1.91 seconds, compared with 18.34 seconds for the full-attention reference. KDA and Mamba-3 incur prefill and decoding overhead (Appendix D.5). The models retain favorable long-context scaling, with overhead depending on the backbone.

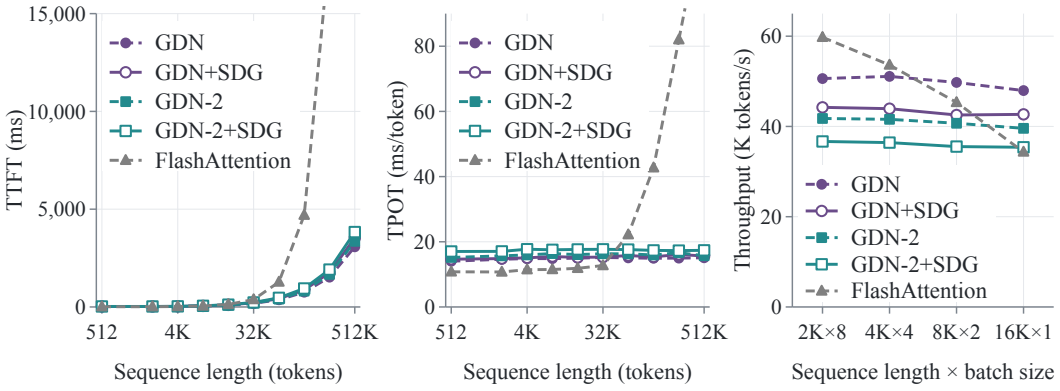


Figure 4: End-to-end efficiency on a single NVIDIA H200. Left and middle: hybrid-model TTFT and TPOT at batch size one. Right: recurrent-only training throughput, with 16,384 input tokens per optimizer step. FlashAttention uses full attention in both comparisons.

6 Conclusion and Limitation

We introduced State-Dependent Gating (SDG), which derives element-wise boundary decay from state changes while preserving native within-chunk computation. Fused implementations support the evaluated backbones. With the realized gate fixed, boundary decay is an element-wise ridge proximal step. Reconstruction analysis quantifies potential gains for ideal gate families under a signal–noise model. In 1.3B models trained on 100B tokens, SDG improves WikiText perplexity and average real-world retrieval across four recurrent backbones. Average retrieval gains extend to three sliding-window hybrids, while zero-shot accuracy and broader long-context results remain mixed.

The gate uses only the net state change and cannot distinguish write–erase histories with the same resulting difference. We evaluate only a 64-token boundary interval, so the effect of other intervals on memory control and parallel efficiency remains untested. Runtime overhead varies across backbones, and cached decoding requires an additional source-state matrix.

USE OF AI ASSISTANCE

Large-language-model tools were used only to polish the writing and improve the clarity of the manuscript. They were not used as an authority for mathematical correctness, to generate experimental measurements, or to replace execution of reported experiments. The authors independently verify all derivations, proofs, citations, code, protocols, numerical results, and final claims, and take full responsibility for the paper.

REPRODUCIBILITY STATEMENT

Appendices C–E give initialization, training, and evaluation settings. Appendices F–I provide implementation details, proofs, and the backward algorithm. We will release the code implementation, model configurations, fused kernels, correctness tests, and training and evaluation scripts.

ETHICS STATEMENT

This work focuses on methodological improvements to linear attention and uses publicly available datasets and benchmarks. We encourage responsible use of the proposed method in downstream applications.

REFERENCES

Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. Language models enable simple systems for generating structured views of heterogeneous data lakes. *arXiv preprint arXiv:2304.09433*, 2023.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Y Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. In *International conference on learning representations*, volume 2024, pp. 15664–15730, 2024a.

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. *arXiv preprint arXiv:2407.05483*, 2024b.

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. *Advances in neural information processing systems*, 29, 2016.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

JT Barron. Squareplus: A softplus-like algebraic rectifier. *arXiv preprint arXiv:2112.11687*, 2021.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *Advances in Neural Information Processing Systems*, 37:107547–107603, 2024.

Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. It's all connected: A journey through test-time memorization, attentional bias, retention, and online optimization. In *International Conference on Learning Representations*, volume 2026, pp. 131306–131333, 2026a.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *Advances in Neural Information Processing Systems*, 38:113506–113543, 2026b.

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.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

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. *arXiv preprint arXiv:2607.07386*, 2026.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

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.

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

Wanyun Cui. A hippocampus for linear attention: An exact memory for what the recurrent state forgets. *arXiv preprint arXiv:2607.02303*, 2026.

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

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.

John Duchi and Yoram Singer. Efficient online and batch learning using forward backward splitting. *The Journal of Machine Learning Research*, 10:2899–2934, 2009.

John C Duchi, Shai Shalev-Shwartz, Yoram Singer, and Ambuj Tewari. Composite objective mirror descent. In *Colt*, volume 10, pp. 14–26, 2010.

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. The language model evaluation harness, 07 2024. URL `https://zenodo.org/records/12608602`.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

Qiang Hao, Rui Cai, Yanwei Pang, and Lei Zhang. From one tree to a forest: a unified solution for structured web data extraction. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pp. 775–784, 2011.

Ali Hatamizadeh, Yejin Choi, and Jan Kautz. Gated deltanet-2: Decoupling erase and write in linear attention. *arXiv preprint arXiv:2605.22791*, 2026.

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? *arXiv preprint arXiv:2404.06654*, 2024.

Jiaxi Hu, Yongqi Pan, Jusen Du, Disen Lan, Tang Tang, Qingsong Wen, Yuxuan Liang, and Weigao Sun. Improving bilinear rnn with closed-loop control. *Advances in Neural Information Processing Systems*, 38:107480–107506, 2026.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Block-recurrent transformers. *Advances in neural information processing systems*, 35:33248–33261, 2022.

Mandar Joshi, Eunsol Choi, Daniel S 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.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Teuvo Kohonen. Correlation matrix memories. *IEEE transactions on computers*, 100(4):353–359, 1972.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Aakash Sunil Lahoti, Kevin Li, Berlin Chen, Caitlin Wang, Aviv Bick, Zico Kolter, Tri Dao, and Albert Gu. Mamba-3: Improved sequence modeling using state space principles. In *International Conference on Learning Representations*, volume 2026, pp. 86173–86202, 2026.

Zhixuan Lin, Evgenii Nikishin, Xu He, and Aaron Courville. Forgetting transformer: Softmax attention with a forget gate. In *International Conference on Learning Representations*, volume 2025, pp. 69704–69738, 2025.

Bo Liu, Rui Wang, Lemeng Wu, Yihao Feng, Peter Stone, and Qiang Liu. Longhorn: State space models are amortized online learners. In *International Conference on Learning Representations*, volume 2025, pp. 95419–95434, 2025.

Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. Openceres: When open information extraction meets the semi-structured web. 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. 3047–3056, 2019.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL `https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu`.

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

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

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

Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in optimization*, 1(3): 127–239, 2014.

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, et al. Rwkv-7" goose" with expressive dynamic state evolution. *arXiv preprint arXiv:2503.14456*, 2025.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. Random feature attention. *arXiv preprint arXiv:2103.02143*, 2021.

Zhen Qin, Weigao Sun, Dong Li, Xuyang Shen, Weixuan Sun, and Yiran Zhong. Lightning attention-2: A free lunch for handling unlimited sequence lengths in large language models. *arXiv preprint arXiv:2401.04658*, 2024a.

Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. Hgrn2: Gated linear rnns with state expansion. *arXiv preprint arXiv:2404.07904*, 2024b.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

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, 2018.

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

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.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pp. 9355–9366. PMLR, 2021.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazzi. Deltaproduct: Improving state-tracking in linear rnns via householder products. *Advances in Neural Information Processing Systems*, 38:153738–153782, 2026a.

Julien Siems, Riccardo Grazzi, Korbinian Pöppel, Jaisidh Singh, Arber Zela, Timur Carstensen, Jenia Jitsev, Frank Hutter, Volkan Cevher, Antonio Orvieto, et al. Complex kda: Understanding and enhancing the expressivity of kimi delta attention. *arXiv preprint arXiv:2609.24797*, 2026b.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

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

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

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

Neehal Tumma, Noel Loo, and Daniela Rus. Preconditioned deltanet: Curvature-aware sequence modeling for linear recurrences. *arXiv preprint arXiv:2604.21100*, 2026.

Johannes von Oswald, Nino Scherrer, Seijin Kobayashi, Luca Versari, Songlin Yang, Maximilian Schlegel, Kaitlin Maile, Yanick Schimpf, Oliver Sieberling, Alexander Meulemans, et al. Mesanet: Sequence modeling by locally optimal test-time training. In *International Conference on Learning Representations*, volume 2026, pp. 26397–26447, 2026.

Ke Alexander Wang, Jiaxin Shi, and Emily B Fox. Test-time regression: a unifying framework for designing sequence models with associative memory, 2025. *URL https://arxiv. org/abs/2501.12352*.

Kaiyue Wen, Xingyu Dang, and Kaifeng Lyu. Rnns are not transformers (yet): The key bottleneck on in-context retrieval. In *International Conference on Learning Representations*, volume 2025, pp. 48813–48856, 2025.

Bernard Widrow and Marcian E. Hoff, Jr. Adaptive switching circuits. In *IRE WESCON Convention Record*, volume 4, pp. 96–104, 1960.

Norbert Wiener. *Extrapolation, interpolation, and smoothing of stationary time series: with engineering applications*. The MIT press, 1949.

Songlin Yang and Yu Zhang. Fla: A triton-based library for hardware-efficient implementations of linear attention mechanism, January 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. *arXiv preprint arXiv:2312.06635*, 2023.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *Advances in neural information processing systems*, 37: 115491–115522, 2024.

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

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.

Yu Zhang, Songlin Yang, Ruijie Zhu, Yue Zhang, Leyang Cui, Yiqiao Wang, Bolun Wang, Freda Shi, Bailin Wang, Wei Bi, et al. Gated slot attention for efficient linear-time sequence modeling. *arXiv preprint arXiv:2409.07146*, 2024.

A Related Work

Recurrent models and forgetting. Linear attention (Katharopoulos et al., 2020) admits recurrent computation; random-feature variants approximate softmax attention (Choromanski et al., 2020; Peng et al., 2021). S4 (Gu et al., 2021) and Mamba (Gu & Dao, 2023) use structured and input-dependent state-space dynamics, respectively. Mamba-3 (Lahoti et al., 2026) adds exponential-trapezoidal discretization and complex-valued transitions. RetNet (Sun et al., 2023), GLA (Yang et al., 2023), and Mamba-2 (Dao & Gu, 2024) support efficient chunkwise computation.

Gated memory designs include outer-product updates in HGRN2 (Qin et al., 2024b), matrix memory in xLSTM's mLSTM (Beck et al., 2024), and slot-based reading in Gated Slot Attention (Zhang et al., 2024). DeltaNet (Schlag et al., 2021; Yang et al., 2024) corrects stored associations using the current state. GDN (Yang et al., 2025) and KDA (Team et al., 2025) add per-head scalar and per-key-channel decay, respectively; GDN-2 (Hatamizadeh et al., 2026) decouples erase and write gates. Their gate predictors use token representations without directly reading the same layer's recurrent memory. RWKV-7 (Peng et al., 2025) generalizes the delta rule with vector-valued gates and in-context learning rates. Forgetting Transformer (Lin et al., 2025) introduces data-dependent decay into softmax attention.

State feedback and transition expressivity. Comba (Hu et al., 2026) combines a closed-loop control interpretation, scalar-plus-low-rank transitions, state and output corrections, and efficient chunkwise computation. Its state correction reads memory, while its scalar forget gate is predicted from the token representation. DeltaProduct (Siems et al., 2026a) composes multiple generalized Householder transformations per token to increase state-tracking expressivity. Complex KDA (Siems et al., 2026b) extends the ranges of KDA's channel-wise gate and delta coefficient to represent rotations while retaining diagonal-plus-rank-one transitions. SDG uses the change in the same layer's memory across a chunk to predict element-wise boundary decay, retaining the supported backbones' native token gates and within-chunk update rules.

Memory capacity and retrieval. Empirical recall gaps (Arora et al., 2024a) and finite-precision bounds for retrieval tasks (Wen et al., 2025) motivate better use of recurrent memory. Block-Recurrent Transformers (Hutchins et al., 2022) use attention and gated state updates across blocks, while Compressive Transformers (Rae et al., 2019) store older activations in a separate compressed memory. More recently, Sparse Delta Memory (Cabannes et al., 2026) expands GDN's state through sparse addressing, and HOLA (Cui, 2026) augments a delta-rule state with a bounded exact key–value cache, using update residuals to select stored associations. These approaches increase or supplement memory capacity. SDG changes retention within the native state matrix, keeping its dimensions fixed and adding no retrieval cache.

Online learning and memory optimization. Matrix associative memories store outer-product superpositions (Kohonen, 1972). Fast-weight programmers learn context-dependent weight changes (Schmidhuber, 1992), while decaying fast-weight memories retain recent activity (Ba et al., 2016). Delta-rule models, test-time training, and Longhorn connect recurrent updates to online learning (Schlag et al., 2021; Sun et al., 2024; Liu et al., 2025). Test-time regression (Wang et al.) unifies sequence models through regression weights, function classes, and optimization algorithms. Titans (Behrouz et al., 2026b) combines gradient-based memory updates with momentum and forgetting. MesaNet (von Oswald et al., 2026) uses a chunkwise implementation and a conjugate-gradient solver for an in-context regression objective; Preconditioned DeltaNet (Tumma et al., 2026) introduces diagonal curvature corrections to delta-rule models with efficient chunkwise algorithms. These methods refine memory learning, while SDG adds a learned retention step after the base chunk update.

Retention as regularization. MIRAS (Behrouz et al., 2026a) separates associative-memory objectives, retention regularization, and online optimization, and derives alternative forgetting mechanisms. Consistent with this connection between forgetting and regularization, SDG boundary decay is an element-wise ridge proximal step when the realized gate is held fixed (Duchi & Singer, 2009; Duchi et al., 2010; Parikh & Boyd, 2014). Its penalty coefficients depend on chunk-level state changes, and shrinkage is applied at chunk boundaries. Wiener shrinkage (Wiener, 1949) informs our oracle reconstruction analysis of adaptive and shared boundary gates.

B NOTATION

Table 5: Symbols used throughout the paper. Shapes are per head unless stated otherwise.

Symbol	Shape or type	Meaning
t, n	indices	token and chunk indices
H	scalar	number of recurrent heads
d_k, d_v	scalars	key/query and value/output dimensions
d_s	scalar	Mamba-3 SSM state width per head
C	scalar	chunk size
$\mathbf{Q}_{[n]}, \mathbf{K}_{[n]}, \mathbf{V}_{[n]}$	chunk tensors	query, key, and value blocks of chunk n
$\mathbf{S}^{+}_{[n]}$	$d_v \times d_k$	state entering chunk n
$\mathbf{S}^{-}_{[n+1]}$	$d_v \times d_k$	raw state after chunk n
$\mathbf{S}^{+}_{[n+1]}$	$d_v \times d_k$	state after SDG
$\Delta\mathbf{S}_{[n]}$	$d_v \times d_k$	state change over chunk n
$\mathbf{W}_S$	$d_k \times d_k$	learned key-channel projection
$\mathbf{X}_{[n]}$	$d_v \times d_k$	projected state change
$\mathbf{G}$	$d_v \times d_k$	learned element-wise gain
$\mathbf{Z}_{[n]}$	$d_v \times d_k$	signed decay score
ρ_0	scalar	fixed positive base rate
$\boldsymbol{\rho}_{[n]}$	$d_v \times d_k$	non-negative boundary rate
$\boldsymbol{\Lambda}_{[n]}$	$d_v \times d_k$	element-wise decay gate; $(0, 1]$ in Eq. (3), $(0, 1)$ under Eq. (6)
$\mathcal{F}$	operator	original layer's chunk recurrence
$\mathcal{G}_\theta$	operator	SDG gate function with learned parameters θ
Analysis (Section 4)		
$\boldsymbol{\kappa}$	$d_v \times d_k$	ridge penalty weights, $\boldsymbol{\kappa} = e^{\boldsymbol{\rho}} - \mathbf{1}$
$\mathbf{S}^\star, \mathbf{N}$	$d_v \times d_k$	target state and residual interference, $\mathbf{S}^- = \mathbf{S}^\star + \mathbf{N}$
Φ	random element	oracle conditioning information, e.g. $(\mathbf{S}^{+}_{[n]}, \mathbf{S}^{-}_{[n+1]})$
$\mathbf{m}_\star, \mathbf{m}_N, \mathbf{m}_-$	$d_v \times d_k$	element-wise second moments given Φ; $\mathbf{m}_- = \mathbf{m}_\star + \mathbf{m}_N$
$\bar{\mathbf{m}}$	$d_v \times d_k$	unconditional second moment $\mathbb{E}[\mathbf{m}_-]$
$\mathcal{R}$	functional	reconstruction risk $\mathbb{E}\|\boldsymbol{\Lambda} \odot \mathbf{S}^- - \mathbf{S}^\star\|_F^2$
$\boldsymbol{\Lambda}^\star, \bar{\boldsymbol{\Lambda}}$	$d_v \times d_k$	optimal Φ-measurable gate and optimal constant gate
$\mathcal{E}_{\text{state}}, \mathcal{E}_{\text{value}}, \mathcal{E}_{\text{key}}$	scalars	excess risks in Eq. (11)

C GATE PARAMETERS AND INITIALIZATION

The gate function in Section 3.2 learns a projection $\mathbf{W}_S$ and an element-wise gain $\mathbf{G}$ for each head. This adds

$$H(d_k^2 + d_v d_k) \tag{12}$$

parameters to each SDG-enabled linear attention layer. We initialize

$$\mathbf{G} = \mathbf{0}, \qquad \rho_0 = -\log \lambda_0, \qquad \lambda_0 = 0.9999. \tag{13}$$

This gives $\boldsymbol{\Lambda}_{[n]} = \lambda_0 \mathbf{1}$ at every boundary before training, independently of the state and projection. Each head's $\mathbf{W}_S$ uses Xavier uniform initialization with bounds $\pm\sqrt{3/d_k}$. Both $\mathbf{W}_S$ and $\mathbf{G}$ are stored in FP32. They are trainable, while ρ_0 is fixed throughout training. At initialization, the zero gain makes the gradient of $\mathbf{W}_S$ zero. The projection begins to learn after $\mathbf{G}$ changes.

D EXPERIMENTAL SETTINGS

D.1 MODELS

Table 6 reports parameter counts. The counts include normalization parameters and all additional SDG parameters. The models have 18 layers and hidden width 2304. GDN, GDN-2, and KDA use

head dimension 128. The Mamba-3 experiments use the SISO variant. Hybrid models replace every other recurrent layer with sliding-window attention and use a window of 2048 tokens.

Table 6: Non-embedding parameter counts for recurrent models, excluding the input embedding and language-model head. SDG is added without reducing the width of the corresponding base model.

Model	Base	With SDG	Added
GDN	1,302,419,520	1,311,856,704	9,437,184
KDA	1,300,649,760	1,310,086,944	9,437,184
Mamba-3	1,303,308,576	1,335,159,072	31,850,496
GDN-2	1,302,640,416	1,312,077,600	9,437,184

D.2 Training

The models are trained for 100B tokens using sequences of 4096 tokens and a global batch of 512 sequences. We train with AdamW, $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1, and gradient clipping at 1.0. The peak learning rate is 4×10^{-4}. The learning rate follows a linear warm-up and cosine decay, with a warm-up over 1B tokens and a final rate of one tenth of the peak rate. Training uses bfloat16.

Each baseline and its +SDG counterpart use the same data order and optimizer settings. The initial boundary decay factor is $\lambda_0 = 0.9999$, corresponding to the base rate $\rho_0 = -\log \lambda_0$ in Eq. (13).

D.3 Evaluation

We evaluate language modeling and zero-shot reasoning using the Language Model Evaluation Harness (Gao et al., 2024). Language modeling performance is measured by perplexity on WikiText (Merity et al., 2016) and LAMBADA (Paperno et al., 2016). The zero-shot suite comprises LAMBADA, PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-Easy, ARC-Challenge (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018), SocialIQA (Sap et al., 2019), and BoolQ (Clark et al., 2019). We report accuracy on these tasks, with length normalization for HellaSwag and ARC-Challenge.

For synthetic retrieval, we evaluate S-NIAH-1 and S-NIAH-2 at 1K, 2K, 4K, and 8K tokens, and S-NIAH-3 and MK-NIAH-1 at 1K, 2K, and 4K tokens. Real-world retrieval is evaluated on SWDE (Hao et al., 2011; Lockard et al., 2019), FDA (Arora et al., 2023), SQuAD (Rajpurkar et al., 2018), DROP (Dua et al., 2019), Natural Questions (Kwiatkowski et al., 2019), and TriviaQA (Joshi et al., 2017). We assess long-context understanding on all 21 LongBench (Bai et al., 2024) tasks, reporting category means and the unweighted average over tasks in Table 10.

D.4 Efficiency protocol

Training throughput. We measure end-to-end training throughput for the recurrent-only models on a single NVIDIA H200. Each step processes 16,384 input tokens, using sequence-length $\times$ batch-size configurations of 2K$\times$8, 4K$\times$4, 8K$\times$2, and 16K$\times$1. We report median throughput after warm-up in thousands of input tokens per second (K tokens/s). Each timed step includes gradient reset, the forward pass and loss computation, backpropagation, gradient clipping, and the AdamW update. The optimizer updates FP32 master parameters and copies the updated weights to the model. Models receive identical inputs. We use neither distributed training nor gradient accumulation.

Inference latency. We measure end-to-end inference latency on a single NVIDIA H200 with batch size one. After warm-up, we run five TTFT trials and three TPOT trials. TTFT is the synchronized elapsed time for prompt processing and first-token prediction. TPOT is the average time per token over 256 subsequent cached greedy decoding steps. We report the median across trials for each metric.

D.5 Detailed efficiency results

Tables 7 and 8 give the complete median inference latencies for the four backbones and the FlashAttention reference, including the GDN and GDN-2 curves in Figure 4. These inference runs use hybrid models with alternating recurrent and sliding-window attention layers (window size 2048), whereas the training runs in Table 9 use recurrent-only models. FlashAttention uses full attention in both comparisons. All runs use a single NVIDIA H200. Sequence-length labels use $1\mathrm{K} = 1024$ tokens.

Table 7: Time to first token (TTFT, ms). Values are medians over five trials at batch size one. Base and +SDG denote the original and gated hybrid models, respectively. FlashAttn. is the full-attention reference. The - indicates OOM.

Length	GDN		KDA		Mamba-3		GDN-2		FlashAttn.
	Base	+SDG	Base	+SDG	Base	+SDG	Base	+SDG	
512	22.53	24.38	23.84	24.82	16.77	17.88	24.19	26.13	11.76
2K	23.66	24.41	24.00	25.57	18.98	31.44	24.79	25.97	12.13
4K	25.46	29.01	27.04	31.05	34.17	58.66	27.60	31.66	22.68
8K	46.88	53.40	50.24	59.47	65.52	114.29	51.14	59.09	50.21
16K	91.60	105.59	97.46	114.30	130.93	227.73	100.32	117.67	131.11
32K	184.57	210.21	196.31	227.36	259.83	452.57	200.47	230.83	380.85
64K	367.49	424.45	398.35	466.09	534.25	920.80	400.97	466.97	1275.78
128K	755.20	868.70	806.35	948.42	1086.01	1888.16	810.34	945.03	4694.28
256K	1523.78	1750.00	1658.86	1909.11	—	—	1664.77	1912.12	18335.50
512K	3075.18	3529.47	3345.86	3841.77	—	—	3356.33	3829.97	—

Table 8: Time per output token (TPOT, ms/token). Each trial averages 256 cached decoding steps. Entries are medians over three trials at batch size one. Model configurations follow Table 7.

Length	GDN		KDA		Mamba-3		GDN-2		FlashAttn.
	Base	+SDG	Base	+SDG	Base	+SDG	Base	+SDG	
512	14.03	14.65	14.53	15.91	17.99	15.94	15.19	17.03	10.78
2K	14.63	14.84	14.62	16.28	18.49	16.22	15.68	17.05	10.74
4K	14.84	15.07	15.31	16.43	18.76	16.40	15.97	17.72	11.44
8K	14.83	15.47	15.20	16.60	18.80	17.03	16.38	17.54	11.50
16K	14.97	15.21	14.84	16.78	18.67	16.42	16.07	17.65	11.86
32K	15.18	15.31	15.01	16.58	18.91	16.57	16.32	17.69	12.71
64K	15.06	15.77	15.47	16.77	18.80	16.40	16.30	17.61	22.18
128K	14.94	15.36	15.07	16.59	19.15	16.70	15.95	17.35	42.71
256K	14.98	16.11	15.56	16.89	—	—	16.65	17.30	82.00
512K	15.09	15.62	14.87	16.99	—	—	15.72	17.37	—

Table 9: End-to-end training throughput in thousands of input tokens/s (higher is better). Each column specifies sequence length × microbatch size and processes 16,384 input tokens per optimizer step. Parentheses give +SDG throughput as a percentage of its corresponding base model. These are the values used for the training curves in Figure 4, with KDA and Mamba-3 included for completeness.

Model	2K×8	4K×4	8K×2	16K×1
GDN	50.609	51.089	49.751	47.951
GDN+SDG	44.233 (87.4%)	43.936 (86.0%)	42.537 (85.5%)	42.677 (89.0%)
KDA	42.403	42.536	41.797	40.571
KDA+SDG	36.721 (86.6%)	36.155 (85.0%)	35.820 (85.7%)	36.148 (89.1%)
Mamba-3	29.496	28.153	25.600	21.976
Mamba-3+SDG	21.650 (73.4%)	20.664 (73.4%)	19.226 (75.1%)	15.932 (72.5%)
GDN-2	41.768	41.576	40.707	39.579
GDN-2+SDG	36.672 (87.8%)	36.421 (87.6%)	35.537 (87.3%)	35.384 (89.4%)
FlashAttention	59.760	53.617	45.328	34.343

E Additional experimental results

E.1 LongBench results for recurrent models

On LongBench (Bai et al., 2024), SDG raises the Code category score by 1.81–2.62 points across all four backbones (Table 10). Average scores improve for GDN, KDA, and Mamba-3, but fall from 11.65 to 11.15 for GDN-2. Its few-shot score falls from 26.94 to 20.85, outweighing gains in code and question answering. Retrieval improvements thus do not translate uniformly to broader long-context performance.

Table 10: LongBench scores. Columns is category means. Avg. is unweighted means over all 21 tasks.

Model	Code	Few-shot	Multi-doc QA	Single-doc QA	Summ.	Synthetic	Avg.
GDN	23.19	17.20	**6.22**	6.89	10.39	2.86	10.37
GDN+SDG	**25.00**	**19.10**	5.62	**7.24**	**10.86**	**2.87**	**10.95**
KDA	20.05	**23.83**	5.27	5.60	9.74	**1.88**	10.64
KDA+SDG	**22.59**	23.10	**6.03**	**6.37**	**10.34**	1.55	**11.10**
Mamba-3	20.37	18.69	**6.46**	**7.06**	8.62	**2.18**	10.03
Mamba-3+SDG	**22.22**	**20.08**	5.94	6.82	**11.63**	1.28	**10.77**
GDN-2	22.38	**26.94**	5.19	6.65	**10.09**	1.46	**11.65**
GDN-2+SDG	**25.00**	20.85	**6.41**	**7.05**	9.83	**2.53**	11.15

E.2 Hybrid models

Tables 11–14 report results for hybrid models that alternate recurrent layers with sliding-window attention. The evaluation follows the recurrent-model comparisons in Section 5. Bold marks the better result within each baseline–SDG pair, including ties.

Language modeling and reasoning. Table 11 shows that SDG improves LAMBADA perplexity across all three hybrid models. GDN also improves on WikiText and average commonsense reasoning accuracy, while KDA and GDN-2 show small declines on these measures. The language modeling and reasoning benefits are thus less consistent than in the recurrent-only models.

Synthetic retrieval. Table 12 shows that the effect of SDG varies across tasks and input lengths. GDN-2 benefits most broadly, improving multi-key retrieval across all evaluated lengths and single-

Table 11: Language modeling and zero-shot reasoning for hybrid models. Metrics follow Table 1. Avg. is the mean over the nine accuracy columns.

Model	Wiki. ppl ↓	LMB. ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	OBQA acc ↑	SIQA acc ↑	BoolQ acc ↑	Avg. acc ↑
GDN	16.01	11.70	48.03	72.25	43.62	**57.77**	70.24	**35.24**	30.40	**40.89**	59.63	50.90
GDN+SDG	**15.59**	**11.34**	**49.04**	**73.29**	**44.42**	56.91	**72.14**	34.39	**31.00**	40.35	**63.85**	**51.71**
KDA	**15.65**	11.27	48.92	**72.85**	**44.72**	56.83	**71.68**	**37.29**	**32.00**	41.20	**62.60**	**52.01**
KDA+SDG	15.66	**11.05**	**49.76**	72.69	44.48	**57.70**	71.59	35.58	31.60	**41.56**	62.51	51.94
GDN-2	**15.44**	11.59	48.92	**72.31**	**44.55**	**58.33**	69.78	**36.52**	**32.60**	41.15	62.02	**51.80**
GDN-2+SDG	15.60	**11.33**	**49.17**	72.09	43.70	58.01	**70.24**	33.96	31.60	**41.20**	**63.96**	51.55

needle retrieval in several longer-context settings. GDN shows clear gains on S-NIAH-3, while KDA has a more mixed response. As in the recurrent-only models, improvements on synthetic retrieval do not imply uniformly better length extrapolation.

Table 12: RULER retrieval accuracy (%) for hybrid models.

Model	S-NIAH-1				S-NIAH-2				S-NIAH-3			MK-NIAH-1		
	1K	2K	4K	8K	1K	2K	4K	8K	1K	2K	4K	1K	2K	4K
GDN	**100.0**	96.4	**51.4**	**24.4**	**100.0**	**100.0**	**56.4**	**26.6**	92.2	71.8	37.0	77.8	63.4	**44.8**
GDN+SDG	**100.0**	**98.8**	47.2	23.8	**100.0**	99.6	52.6	25.8	**95.0**	**90.4**	**39.4**	**84.6**	**75.0**	42.2
KDA	**100.0**	**100.0**	**54.6**	**26.2**	**100.0**	**100.0**	54.8	29.6	**94.0**	81.6	31.0	83.6	**73.2**	39.4
KDA+SDG	**100.0**	99.4	53.8	**26.2**	**100.0**	**100.0**	**56.4**	**29.8**	91.6	**84.8**	**34.2**	**84.2**	69.4	**40.4**
GDN-2	**100.0**	**100.0**	54.2	22.0	**100.0**	**100.0**	44.6	20.8	98.2	**97.8**	36.4	82.0	78.4	38.6
GDN-2+SDG	**100.0**	**100.0**	**57.8**	**26.2**	**100.0**	99.8	**55.8**	**28.0**	**99.4**	92.0	**41.6**	**86.6**	**81.4**	**42.0**

Real-world retrieval. Table 13 shows higher average retrieval accuracy for all three hybrid models, extending the trend observed in the recurrent-only setting. GDN and KDA show clearer average gains, while GDN-2 changes little overall because improvements and declines across tasks largely offset one another. These results suggest that SDG can complement sliding-window attention in hybrid models.

Table 13: Real-world retrieval accuracy (%) for hybrid models. Avg. is the mean over the six tasks.

Model	SWDE	SQD	FDA	TQA	NQ	DROP	Avg.
GDN	46.39	41.82	59.85	61.20	**25.66**	**23.24**	43.03
GDN+SDG	**48.17**	**44.71**	**60.13**	**62.44**	25.28	23.10	**43.97**
KDA	43.96	**43.37**	58.86	62.03	26.39	**24.68**	43.22
KDA+SDG	**45.08**	43.23	**61.94**	**64.22**	**27.02**	22.95	**44.07**
GDN-2	**49.86**	**43.67**	62.03	**63.45**	25.15	24.25	44.74
GDN-2+SDG	48.92	42.29	**64.49**	62.03	**26.51**	**24.77**	**44.84**

Long-context understanding. Table 14 shows consistent gains in code and few-shot learning across all three hybrid models. Average performance improves for KDA and GDN-2 but declines for GDN, whose gains are outweighed by weaker question answering and summarization. As in the recurrent-only setting, retrieval improvements do not translate uniformly to broader long-context understanding.

Table 14: LongBench scores (%) for hybrid models. Columns give category means. Avg. is the unweighted mean over all 21 tasks. Higher is better.

Model	Code	Few-shot	Multi-doc QA	Single-doc QA	Summ.	Synthetic	Avg.
GDN	19.62	25.64	**7.18**	**7.52**	**11.85**	**2.11**	**12.11**
GDN+SDG	**21.26**	**27.86**	5.70	5.32	9.37	2.02	11.50
KDA	20.10	26.23	**7.18**	**7.05**	9.08	1.36	11.54
KDA+SDG	**21.27**	**26.63**	5.85	6.53	**10.07**	**1.77**	**11.63**
GDN-2	21.39	25.27	6.85	**7.63**	**9.99**	**1.95**	11.79
GDN-2+SDG	**28.43**	**25.74**	**6.87**	6.74	9.47	1.65	**12.24**

E.3 Ablation settings

Gate inputs. The fixed boundary control uses $\mathbf{\Lambda}_{[n]} = 0.9999\mathbf{1}$. The learned static control sets $\mathbf{Z}_{[n]} = \mathbf{G}$ in Eq. (5), with zero-initialized trainable $\mathbf{G}$. These controls examine whether the boundary gate benefits from adapting to its input. The chunk-input control further tests whether directly using the layer's state changes provides additional value. It uses $\mathbf{\Omega}_{[n]}$ from the affine chunk transition in Eq. (79) of Appendix H. For fixed chunk token representations and native data-dependent gates, this is the state produced by the native chunk update from a zero incoming state. In GDN, it is obtained by setting $\mathbf{S}^{+}_{[n]} = \mathbf{0}$ in Eq. (37). It retains the chunk's writes and their subsequent within-chunk updates, while excluding the response of the incoming memory.

The control replaces $\Delta\mathbf{S}_{[n]}$ with $\mathbf{\Omega}_{[n]}$ in Eq. (4) and retains the rate and boundary update in Eqs. (5)–(6). The main recurrence still starts from the actual $\mathbf{S}^{+}_{[n]}$, and the gate scales the entire updated state $\mathbf{S}^{-}_{[n+1]}$. Only the gate input changes. The independence of $\mathbf{\Omega}_{[n]}$ from the incoming state is within the current layer, since its token representations can contain context from earlier layers.

The state-input control replaces $\Delta\mathbf{S}_{[n]}$ with $\mathbf{S}^{+}_{[n]}$ in Eq. (4). It reads the memory entering the chunk and retains the same gate parameterization and boundary update, allowing a comparison between the incoming state and its change across the chunk.

Channel-wise parameterization. The channel-wise variant retains the per-head projection $\mathbf{W}_S \in \mathbb{R}^{d_k \times d_k}$ and gain $\mathbf{G} \in \mathbb{R}^{d_v \times d_k}$ of element-wise SDG. It computes $\mathbf{Z}_{[n]} = \mathbf{G} \odot (\Delta\mathbf{S}_{[n]}\mathbf{W}_S^{\top})$ as in Eq. (4) and averages these scores over value rows before applying the nonlinear rate map:

$$\bar{z}_{[n],b} = \frac{1}{d_v}\sum_{a=1}^{d_v} Z_{[n],ab}, \tag{14}$$

$$\lambda_{[n],b} = \exp\Big(-\bar{z}_{[n],b} - \sqrt{\bar{z}^2_{[n],b} + \rho_0^2}\Big). \tag{15}$$

The resulting coefficient for each key channel is broadcast across all value rows, giving $\Lambda_{[n],ab} = \lambda_{[n],b}$ and $\mathbf{\Lambda}_{[n]} = \mathbf{1}_{d_v}\boldsymbol{\lambda}_{[n]}^{\top}$. This gate scales the entire updated state through Eq. (6). The variant uses the same zero initialization for $\mathbf{G}$ and the same fixed $\rho_0 = -\log\lambda_0$, so all retention coefficients initially equal $\lambda_0 = 0.9999$. It adds $H(d_k^2 + d_v d_k)$ parameters per layer, matching element-wise SDG.

Paired training. Both gate-input controls and SDG use identical initial values for the backbone and $\mathbf{W}_S$, with $\mathbf{G} = \mathbf{0}$ and $\lambda_0 = 0.9999$. They therefore start from the same model function. The comparison holds the data order, training token budget, optimizer, learning-rate schedule, chunk size, gated layers, and evaluation settings fixed. Each variant uses its own gate input throughout training and evaluation. A comparison at a reduced training budget requires training all compared variants at that budget.

Parameter and computation costs. All three dynamic inputs use $H(d_k^2 + d_v d_k)$ additional parameters per layer and the same gate projection. Constructing $\mathbf{\Omega}_{[n]}$ explicitly requires a matrix

multiplication costing approximately $2Cd_vd_k$ forward FLOPs per head and boundary, whereas constructing $\Delta\mathbf{S}_{[n]}$ requires a state subtraction. The state-input control uses the already available $\mathbf{S}^+_{[n]}$. Thus, matching the parameter count and gate function does not imply equal total computation. The identity-projection control retains the state-difference input and element-wise gain while removing the learned projection and its matrix multiplication. The gate-input comparison therefore matches parameter count while allowing different input-construction costs.

Scope of the comparisons. The $\mathbf{\Omega}$-input and incoming-state controls test the input choice with the gate parameterization fixed. The granularity comparison changes coefficient sharing while retaining the same parameter count. The identity-projection control reduces the parameter count by removing $\mathbf{W}_S$. The reported scores are point estimates without uncertainty estimates, so small differences between variants do not establish a statistically reliable advantage.

Gate-input runtime. Table 15 compares the complete GDN operator with $\Delta\mathbf{S}$- and $\mathbf{\Omega}$-input SDG. Timing includes query/key normalization, WY preparation, the fused state scan, and output kernels. Backward includes boundary recomputation and gradients for all inputs and gate parameters. The timings are measured independently, with backward-only runs reusing a fixed forward graph, so their medians need not be additive. At 16,384 tokens per batch, $\mathbf{\Omega}$ increases forward latency by 1.30–6.07%, backward latency by 46.17–49.29%, and combined latency by 35.89–39.17%. At batch size one, the differences are more shape-dependent, with a small forward decrease at 512 tokens. These measurements favor $\Delta\mathbf{S}$ for training-time operator efficiency but do not measure full-model runtime.

Table 15: Median GDN operator latency (ms) on an RTX 5090. Δt is the percentage change for $\mathbf{\Omega}$ relative to $\Delta\mathbf{S}$. The upper block fixes $LB = 16{,}384$ tokens per batch, and the lower block uses $B = 1$.

	Forward			Backward			Forward + backward		
$L \times B$	$\Delta\mathbf{S}$	$\mathbf{\Omega}$	Δt (%)	$\Delta\mathbf{S}$	$\mathbf{\Omega}$	Δt (%)	$\Delta\mathbf{S}$	$\mathbf{\Omega}$	Δt (%)
2048×8	5.164	5.451	+5.56	12.342	18.426	+49.29	15.765	21.939	+39.17
4096×4	5.112	5.422	+6.07	12.867	19.030	+47.90	16.637	22.875	+37.50
8192×2	4.800	4.912	+2.34	13.252	19.513	+47.25	16.850	23.353	+38.60
16384×1	5.705	5.779	+1.30	12.686	18.542	+46.17	16.624	22.589	+35.89
512×1	0.570	0.539	−5.53	2.013	2.298	+14.13	2.180	2.193	+0.59
2048×1	0.850	0.891	+4.80	1.635	2.061	+26.09	2.571	2.946	+14.59
4096×1	1.378	1.443	+4.73	3.321	4.398	+32.42	4.309	5.623	+30.51
8192×1	3.026	3.057	+1.04	6.455	9.567	+48.20	8.800	11.757	+33.60

E.4 Boundary gate distributions

We examine the distribution of the element-wise retention coefficient Λ at observed chunk boundaries. We report the fractional reduction $d = 1 - \Lambda$, so larger values indicate stronger decay. Figure 5 shows the mean reduction across layers and heads. Table 16 reports statistics from the original model archives.

Observations and aggregation. Each model archive contains 18 layers, 32 sample IDs, and eight boundary positions per sample: 64, 640, 1216, 1792, 2368, 2944, 3520, and 4096 tokens consumed. This gives 256 boundary observations per layer. GDN, KDA, and GDN-2 each contribute 67,108,864 gate-cell observations per layer, across 16 heads. For Mamba-3, gate statistics are computed from the first 16 heads. Each measured heatmap cell averages the reduction over all observed samples, boundaries, and state elements for one layer and head. Means and threshold fractions use all observed cells; the reported medians use a uniform reservoir of 65,536 cells per layer. Repeated observations are not treated as independent experimental replicates. The reduction is computed stably as $-\operatorname{expm1}(-\rho)$, retaining small values even when FP32 retention rounds to one.

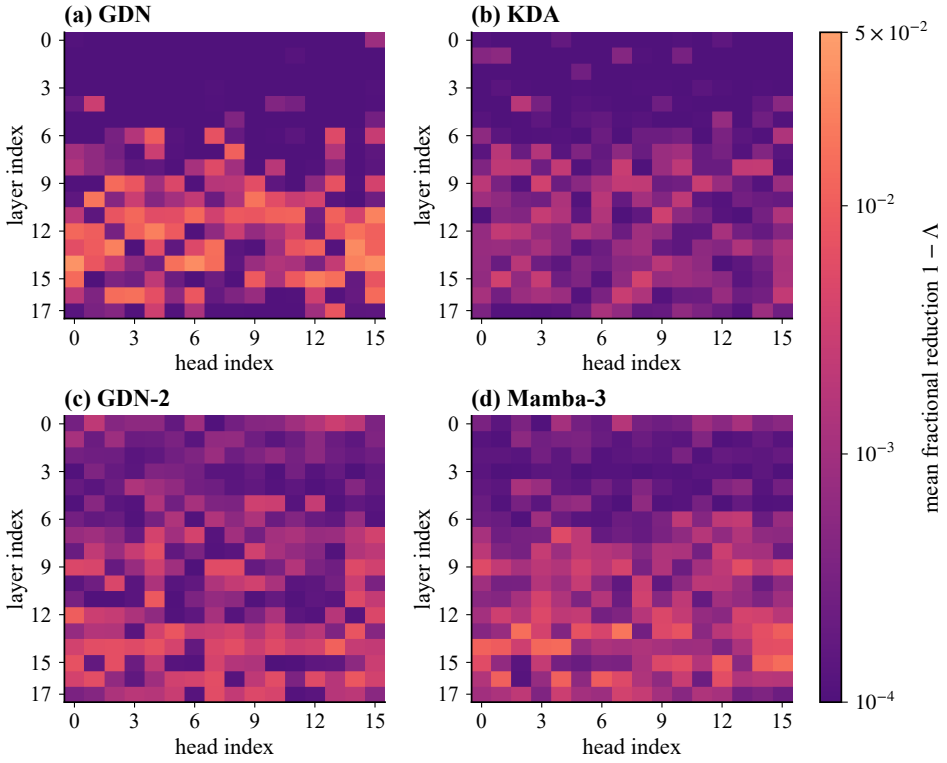


Figure 5: Mean fractional reduction $1 - \Lambda$ across layers and heads on a shared logarithmic color scale. Mamba-3 statistics use the first 16 heads.

Distributions in the delta-rule models. For GDN, KDA, and GDN-2, every layer's sampled median reduction is approximately 10^{-4}, corresponding to retention near the initialization value 0.9999. The means are higher and vary across layers. Averaged over all observed cells, the reduction is 0.3104%, 0.0538%, and 0.1496%, respectively (Table 16). GDN and GDN-2 reach their largest layer means at layer 14 (1.0982% and 0.4693%); KDA peaks at layer 8 (0.1161%). Only 0.69%, 0.06%, and 0.25% of cells, respectively, have retention below 0.9. Thus, large reductions are concentrated in a small fraction of observations even though the median remains close to its initial value.

Table 16: Measured gate statistics pooled across layers with equal weight per observed cell. All entries are percentages; threshold fractions use the archived FP32 comparisons.

Model	Mean $1 - \Lambda$	$\Pr(\Lambda < 0.99)$	$\Pr(\Lambda < 0.9)$
GDN+SDG	0.3104	4.2875	0.6915
KDA+SDG	0.0538	0.8487	0.0587
GDN-2+SDG	0.1496	2.5786	0.2541
Mamba-3	2.8041	99.6959	1.25×10^{-6}

E.5 BOUNDARY OUTPUT STABILITY

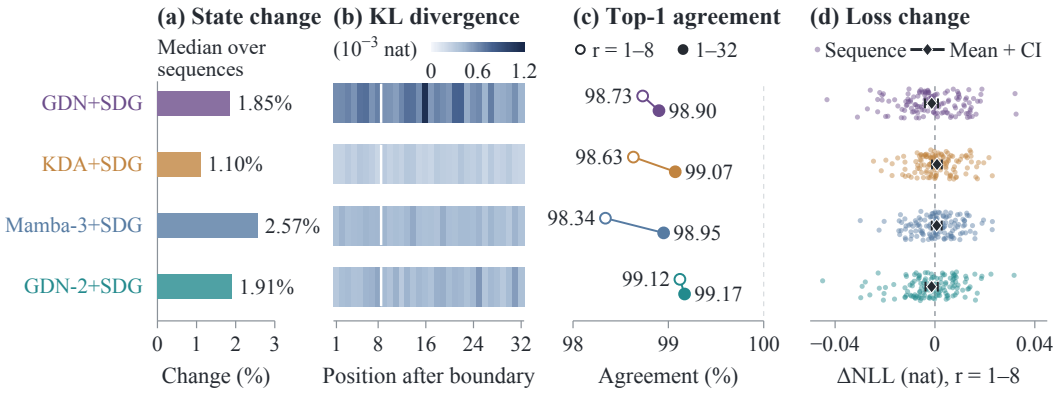


Figure 6: Effects of one SDG boundary update under identical inputs. **(a)** Median relative state change. **(b)** KL divergence from skip to normal. **(c)** Top-1 agreement. **(d)** Sequence-level mean ΔNLL over positions 1–8. Black diamonds and bars denote means and four-model-adjusted bootstrap intervals.

Paired boundary intervention. We compare normal execution with skipping one SDG update in the same trained model. At the selected boundary, we clone the complete cache before applying the gate. The normal branch applies SDG in all recurrent layers, while the skip branch passes the unmodified pre-gate states to the next chunk. All other operations are unchanged. Both branches then consume the same subsequent ground-truth tokens through cached decoding. This comparison isolates the effect of the selected update without changing the input or comparing separately trained models.

Prediction positions and metrics. Let b be the input position after which the selected boundary update occurs, and let r count subsequent input positions. The logits at position $b + r$ predict the ground-truth token y_{b+r+1}. The output at $r = 0$ is computed before the boundary update and is therefore unaffected. We exclude this position from all window averages. Top-1 agreement is the fraction of affected positions at which both branches assign the highest probability to the same token. We report this fraction over $r = 1, \ldots, 8$ and $r = 1, \ldots, 32$, with equal weight for every sequence and position. At each position, we also measure KL divergence from the skip distribution to the normal distribution. This measures a change in the full output distribution rather than only its most probable token. All 32 positions precede the next boundary update.

Prediction loss. Agreement alone does not measure prediction quality, since both branches can predict the same incorrect token. We therefore also measure the negative log-likelihood (NLL) of the ground-truth next token. Given the common token prefix $y_{\leq b+r}$, it is

$$\mathrm{NLL}(r) = -\log p(y_{b+r+1} \mid y_{\leq b+r}),$$

where p is the predictive distribution of the branch under consideration. We use the natural logarithm, so losses are measured in nats. Lower NLL means that the model assigns greater probability to the ground-truth token. We define the loss change as

$$\Delta\mathrm{NLL}(r) = \mathrm{NLL}_{\mathrm{normal}}(r) - \mathrm{NLL}_{\mathrm{skip}}(r). \tag{16}$$

Positive values indicate that applying the update increases the target loss, while negative values indicate a decrease. We average $r = 1, \ldots, 8$ within each sequence before averaging across sequences, and inspect the loss changes at all 32 affected positions.

Samples and controls. The diagnostic includes 128 observations per model. GDN, KDA, and GDN-2 use the same 128 evaluation sequences at training step 47,683, with one selected boundary per sequence. Boundaries at input positions 1024, 2048, and 3072 account for 43, 43, and 42 sequences,

respectively. Each intervention covers all 18 recurrent layers, with 16 heads per layer. For these three models, two unmodified executions produce identical logits, and the logit difference at $r = 0$ is zero. Each layer receives exactly one boundary intervention. Mamba-3 contributes position-wise and sequence-level summaries; cross-model sample matching and intervention controls are assessed for GDN, KDA, and GDN-2.

Prediction agreement. Figure 6 shows that the two branches predict the same token at 98.34–99.12% of the first eight affected positions. Over 32 positions, agreement is 98.90%, 99.07%, 98.95%, and 99.17% for GDN, KDA, Mamba-3, and GDN-2, respectively. These are position-level rates, not the fraction of entire sequences with identical predictions. For GDN, KDA, and GDN-2, all 32 top-1 predictions agree in 70.31%, 71.88%, and 75.00% of the evaluation sequences, respectively. The remaining sequences contain at least one changed prediction. The small KL values complement the agreement rates by showing that the average change in the full distribution is also small.

Uncertainty estimates. We resample whole sequences 5,000 times. Percentile-bootstrap intervals for mean loss changes use Bonferroni correction across four models, with quantiles 0.00625 and 0.99375, targeting approximately 95% simultaneous coverage. The $\epsilon = 0.01$ line in Figure 7 is a descriptive reference for the population mean loss increase, not a per-sequence or per-token bound; our conclusions use the measured confidence bounds. Figure 8 shows pointwise 95% intervals, while the KL heatmap in Figure 3 shows the corresponding means. Window-level agreement percentages are descriptive averages without confidence intervals.

Average loss changes. Table 17 summarizes loss and output changes over the first eight affected positions. All four corrected intervals contain zero, and their upper bounds are below 0.003 nat per token. The small upper bounds constrain the average loss increase under the sampling protocol. The average KL divergence is below 0.00065 nat, while top-1 predictions differ at 0.88–1.66% of the first eight affected positions. Figure 8 retains the full position-wise variation rather than smoothing the curves.

Table 17: Prediction changes over the first eight affected positions. Loss and KL values are in nats. Confidence intervals are adjusted across four models.

Model	Mean ΔNLL	Adjusted interval	Mean KL	Top-1 change
GDN+SDG	-0.001301	$[-0.004024,\ 0.001216]$	0.000640	1.27%
KDA+SDG	0.000834	$[-0.001152,\ 0.002742]$	0.000276	1.37%
Mamba-3+SDG	0.000710	$[-0.001282,\ 0.002695]$	0.000406	1.66%
GDN-2+SDG	-0.001342	$[-0.003787,\ 0.001129]$	0.000399	0.88%

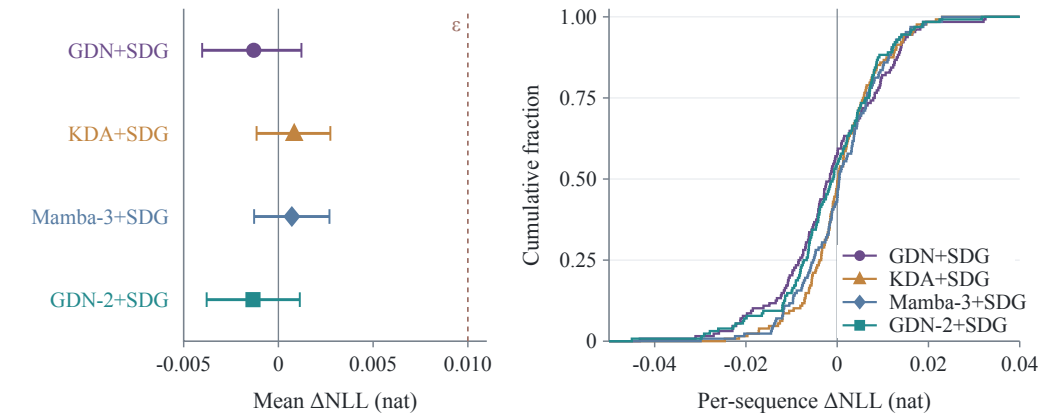


Figure 7: Prediction loss after a single SDG boundary update relative to skipping that update. The left panel reports mean ΔNLL over the first eight affected positions with bootstrap intervals over sequences adjusted across four models for approximately 95% simultaneous coverage. The dotted line marks the reference tolerance $\epsilon = 0.01$ nat per token. The right panel shows the empirical distribution of sequence-level mean changes. Positive values indicate increased loss.

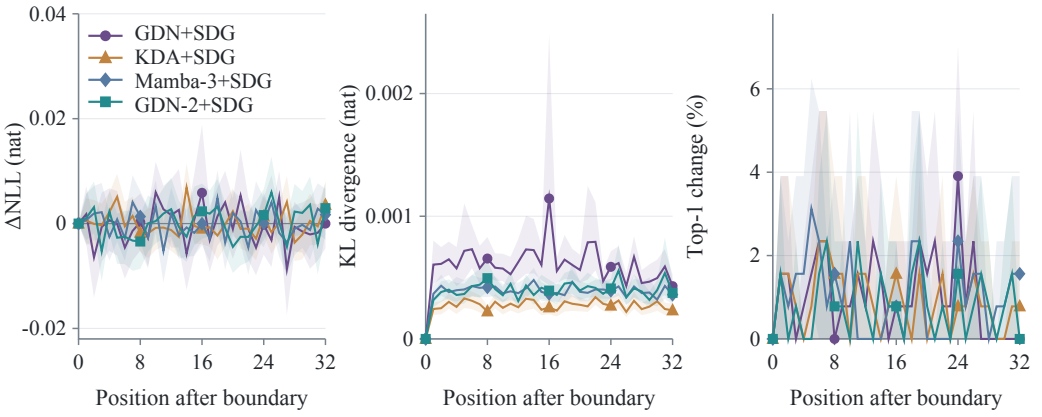


Figure 8: Prediction changes following a single SDG boundary update relative to skipping it. The panels show mean ΔNLL, KL divergence from the skip distribution to the normal distribution, and the fraction of changed top-1 predictions. Shaded regions are pointwise 95% bootstrap intervals over sequences. Position zero is unaffected because its output precedes the update.

Changes to the state. For each head, we measure the relative change caused by the boundary gate as

$$\frac{\left\|\mathbf{S}^{+}_{[n+1]} - \mathbf{S}^{-}_{[n+1]}\right\|_F}{\left\|\mathbf{S}^{-}_{[n+1]}\right\|_F}. \tag{17}$$

This measures the gate's effect on the state, not the change $\Delta\mathbf{S}_{[n]}$ across the chunk used to compute the gate. Figure 9 averages Eq. (17) over heads and sequences for each layer. Its scatter plot relates each sequence's mean change across layers and heads to its mean loss change over the first eight affected positions. The median state changes across sequences, shown in Figure 6, are approximately 1.85%, 1.10%, 2.57%, and 1.91% for GDN, KDA, Mamba-3, and GDN-2, respectively.

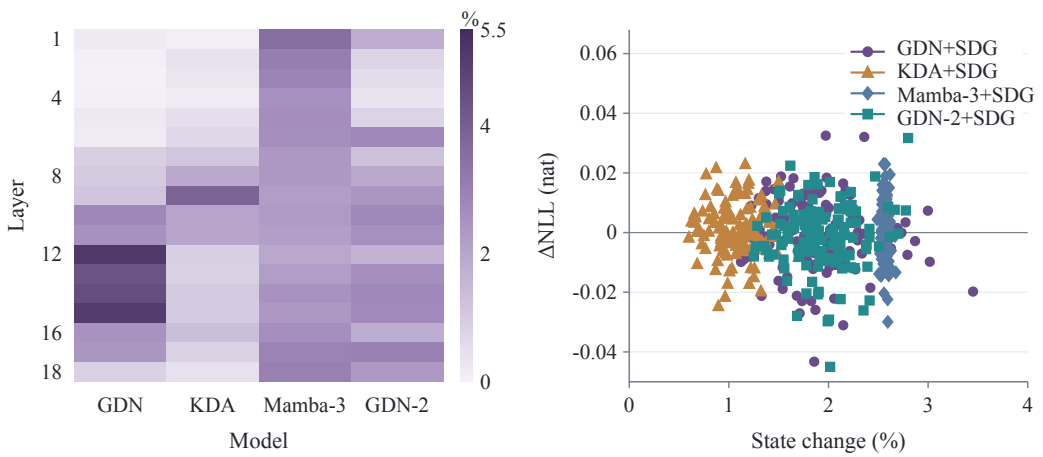


Figure 9: State changes at chunk boundaries. Left: relative Frobenius change per layer, averaged over heads and sequences, on a common scale. Right: sequence-level mean state change versus mean ΔNLL over the first eight affected positions.

Individual loss increases. The 95th percentiles of sequence-level mean loss changes range from 0.0147 to 0.0161 nat across the four models. Individual-token analysis covers GDN, KDA, and GDN-2. Among GDN's 4,096 affected positions, one has a loss increase of 0.656645 nat, occurring 16 positions after the boundary at input position 3072. The normal and skip losses are 1.864409 and 1.207765, respectively, so the normal branch assigns the target about 51.9% of the probability assigned by the skip branch. Both top-1 predictions are the same and incorrect. This event contributes to the response at position 16 in Figure 8, outside the primary eight-position window. Its sequence-level mean state change is about 1.97%, so an unusually large average state change does not identify it. KDA and GDN-2 have no positions with a loss increase above 0.5 nat in this sample.

Boundary-offset diagnostic. We insert valid filler tokens into benchmark inputs to place the start of answer generation before and after chunk boundaries. As specified in Appendix F.9, these tokens enter the native recurrence and advance the within-chunk token counter. They therefore shift answer generation relative to the C-token boundary schedule. Across the tested benchmarks, absolute changes in accuracy are at most 0.5 percentage points on commonsense reasoning and 1 percentage point on retrieval. These are observed changes in benchmark accuracy, measured in percentage points rather than relative percentages. Because the filler tokens also change the input sequence, this diagnostic measures sensitivity to the combined change in input and boundary alignment. It complements the paired boundary intervention above, which holds the token sequence fixed and skips one boundary update.

F State Updates and Implementation

F.1 Extended Preliminaries

State convention. Equation (1) uses value channels as state rows and key channels as columns. The output reads the state after the current write, so it includes the current token. At a fresh sequence start, $\mathbf{S}_0 = \mathbf{0}$. The state size is independent of sequence length L.

Native token updates. **DeltaNet** (Schlag et al., 2021; Yang et al., 2024) corrects the value stored at the current key using a gradient step (Widrow & Hoff, 1960) on the local regression loss

$$\ell_t(\mathbf{S}) = \frac{1}{2}\|\mathbf{S}\boldsymbol{k}_t - \boldsymbol{v}_t\|_2^2. \tag{18}$$

Starting from $\mathbf{S}_{t-1}$, a gradient step with data-dependent step size $\beta_t \in [0, 1]$ gives

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \beta_t(\boldsymbol{v}_t - \mathbf{S}_{t-1}\boldsymbol{k}_t)\boldsymbol{k}_t^\top = \mathbf{S}_{t-1}(\mathbf{I} - \beta_t\boldsymbol{k}_t\boldsymbol{k}_t^\top) + \beta_t\boldsymbol{v}_t\boldsymbol{k}_t^\top. \tag{19}$$

For unit-norm keys, β_t controls how much of the stored value is replaced by $\boldsymbol{v}_t$.

Gated DeltaNet (GDN) (Yang et al., 2025) applies a data-dependent decay gate $\alpha_t \in (0, 1]$ before the delta update:

$$\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top) + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^\top. \tag{20}$$

The scalar α_t sets one retention rate for all state elements within a head.

Kimi Delta Attention (KDA) (Team et al., 2025) assigns a decay coefficient to each key channel. Let $\mathbf{D}_t = \mathrm{Diag}(\boldsymbol{\alpha}_t)$ with $\boldsymbol{\alpha}_t \in (0, 1]^{d_k}$. The state update becomes

$$\mathbf{S}_t = \mathbf{S}_{t-1}\mathbf{D}_t(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top) + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^\top. \tag{21}$$

Each decay coefficient is shared across value channels, while β_t controls both erasing and writing.

Gated DeltaNet-2 (GDN-2) (Hatamizadeh et al., 2026) replaces the shared delta step size β_t with separate erase and write gates, retaining the decay matrix $\mathbf{D}_t$. Let $\boldsymbol{e}_t = \boldsymbol{b}_t \odot \boldsymbol{k}_t$ and $\boldsymbol{z}_t = \boldsymbol{w}_t \odot \boldsymbol{v}_t$, where $\boldsymbol{b}_t \in [0, 1]^{d_k}$ and $\boldsymbol{w}_t \in [0, 1]^{d_v}$ are the erase and write gates. Its update is

$$\mathbf{S}_t = \mathbf{S}_{t-1}\mathbf{D}_t(\mathbf{I} - \boldsymbol{e}_t \boldsymbol{k}_t^\top) + \boldsymbol{z}_t \boldsymbol{k}_t^\top. \tag{22}$$

The erase gate weights the key channels used to read stored content, and the write gate scales the incoming value channels. Setting $\boldsymbol{b}_t = \beta_t \mathbf{1}_{d_k}$ and $\boldsymbol{w}_t = \beta_t \mathbf{1}_{d_v}$ recovers KDA. Further setting $\mathbf{D}_t = \alpha_t \mathbf{I}$ recovers GDN.

In these layers, the gate coefficients are predicted from token representations without directly reading the recurrent state; the delta correction itself reads that state. For fixed token inputs, every value row is propagated through the same transition in key space. This shared structure supports the chunkwise computation in Section 2.2.

The Mamba-3 SISO backbone (Lahoti et al., 2026) has a matrix state and no delta read-and-correct term. Its state-space recurrence and our boundary gating are detailed in Appendix F.8.

Additive chunk computation. We use the complete-chunk convention stated in Section 2.2. The chunkwise kernels considered here combine a recurrent state scan across chunks with parallel matrix operations within each chunk (Yang et al., 2023; Qin et al., 2024a; Yang et al., 2024; 2025; Team et al., 2025; Hatamizadeh et al., 2026). Chunk $n \in \{1, \ldots, N\}$ contains tokens $(n-1)C + 1, \ldots, nC$, with incoming state $\mathbf{S}_{[n]} = \mathbf{S}_{(n-1)C}$. Let $\mathbf{Q}_{[n]}, \mathbf{K}_{[n]} \in \mathbb{R}^{C \times d_k}$ and $\mathbf{V}_{[n]}, \mathbf{O}_{[n]} \in \mathbb{R}^{C \times d_v}$ stack the corresponding token vectors as rows. In the additive case of Eq. (1),

$$\mathbf{S}_{[n+1]} = \mathbf{S}_{[n]} + \mathbf{V}_{[n]}^\top \mathbf{K}_{[n]}, \tag{23}$$

$$\mathbf{O}_{[n]} = \mathbf{Q}_{[n]}\mathbf{S}_{[n]}^\top + \left(\mathbf{Q}_{[n]}\mathbf{K}_{[n]}^\top \odot \mathbf{M}_C\right)\mathbf{V}_{[n]}, \tag{24}$$

where $\mathbf{M}_C$ is a binary lower-triangular mask including the diagonal. The two output terms read the incoming memory and combine causal interactions within the chunk. For fixed chunk size and head dimensions, the arithmetic work scales linearly with sequence length.

Shared factors and the incoming-memory response. Gated delta-rule layers combine cumulative decay with compact WY representations of the accumulated rank-one updates (Bischof & Van Loan, 1987; Yang et al., 2024; 2025; Team et al., 2025; Hatamizadeh et al., 2026). Their factors are shared across value rows and can be computed from token representations before the state scan. Let $\mathbf{P}_{[n]} \in \mathbb{R}^{d_k \times d_k}$ denote the shared chunk transition and $\boldsymbol{\Omega}_{[n]} \in \mathbb{R}^{d_v \times d_k}$ the state produced from zero incoming memory. The resulting affine update and state difference are given in Eq. (2). For fixed token representations and native gates, $\mathbf{P}_{[n]}$ and $\boldsymbol{\Omega}_{[n]}$ do not depend on this layer's incoming state. The additive case has $\mathbf{P}_{[n]} = \mathbf{I}$, so its state change contains only new writes. In the gated layers, the change also reflects how the chunk transforms the existing memory. Appendix F.3 gives the exact chunk formulas.

Training and prefill use chunkwise computation; decoding applies the native recurrence one token at a time. Section 3 uses the incoming and updated chunk states to compute an additional boundary gate.

F.2 CHUNKWISE NOTATION AND BOUNDARY GATING

We present each backbone in the WY/UT matrix form used by GDN (Yang et al., 2025), KDA (Team et al., 2025), and GDN-2 (Hatamizadeh et al., 2026). We retain the state orientation of this paper, $\mathbf{S} \in \mathbb{R}^{d_v \times d_k}$, so the KDA and GDN-2 state equations are transposed relative to their original presentations. For one head and complete chunk n, $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{C \times d_k}$ and $\mathbf{V} \in \mathbb{R}^{C \times d_v}$ stack the native token vectors. We suppress $[n]$ on these chunk-local quantities and their WY factors, but retain it on the input state, output block, and state passed to the next chunk. Let $\mathrm{tril}(\cdot)$ include the diagonal and $\mathrm{tril}(\cdot, -1)$ exclude it; $\oslash$ denotes element-wise division.

In every model, $\mathbf{S}^{+}_{[n]}$ enters the original chunk computation, which produces $\mathbf{O}_{[n]}$ and the raw state $\mathbf{S}^{-}_{[n+1]}$. SDG then computes

$$\begin{aligned} \Delta\mathbf{S}_{[n]} &= \mathbf{S}^{-}_{[n+1]} - \mathbf{S}^{+}_{[n]}, \\ \mathbf{Z}_{[n]} &= \mathbf{G} \odot (\Delta\mathbf{S}_{[n]}\mathbf{W}_S^{\top}), \\ \mathbf{\Lambda}_{[n]} &= \exp\left[-\mathbf{Z}_{[n]} - \sqrt{\mathbf{Z}_{[n]} \odot \mathbf{Z}_{[n]} + \rho_0^2 \mathbf{1}}\right]. \end{aligned} \tag{25}$$

We display the final update $\mathbf{S}^{+}_{[n+1]} = \mathbf{\Lambda}_{[n]} \odot \mathbf{S}^{-}_{[n+1]}$ alongside each backbone's chunk equations below. In all four cases SDG changes only the matrix carried to the next chunk: the gate acts on the entire raw state including the chunk's writes, whereas the WY factors, the pseudo-value term, the intra-chunk product, the output block, and the native query scaling, normalization, output gates, and output projections are untouched. The WY factors are computed from token-local inputs, while $\mathbf{\Lambda}_{[n]}$ is evaluated inside the state scan. We do not repeat this per backbone below.

F.3 DERIVATION OF THE CHUNK CLOSED FORMS

The closed forms in Appendices F.5–F.7 are three specializations of one lemma. Within a complete chunk, stack $\boldsymbol{e}_i^{\top}$ into $\mathbf{E} \in \mathbb{R}^{C \times d_k}$ and $\boldsymbol{z}_i^{\top}$ into $\mathbf{Z} \in \mathbb{R}^{C \times d_v}$ (chunk-local matrices; this $\mathbf{Z}$ is not the SDG score $\mathbf{Z}_{[n]}$), and let $\boldsymbol{\gamma}_i = \bigodot_{j \le i} \boldsymbol{\alpha}_j$ with the convention $\boldsymbol{\gamma}_0 = \mathbf{1}$. Define the cumulative and remaining decay matrices $\mathbf{\Gamma}^{\mathrm{pre}}, \mathbf{\Gamma}^{\mathrm{tail}} \in \mathbb{R}^{C \times d_k}$ row-wise by

$$\mathbf{\Gamma}^{\mathrm{pre}}_{i,:} = \boldsymbol{\gamma}_i^{\top}, \qquad \mathbf{\Gamma}^{\mathrm{tail}}_{i,:} = (\boldsymbol{\gamma}_C \oslash \boldsymbol{\gamma}_i)^{\top}, \tag{26}$$

together with

$$\bar{\mathbf{K}} = \mathbf{K} \oslash \mathbf{\Gamma}^{\mathrm{pre}}, \quad \bar{\mathbf{E}} = \mathbf{\Gamma}^{\mathrm{pre}} \odot \mathbf{E}, \quad \mathbf{A} = \left[\mathbf{I}_C + \mathrm{tril}(\bar{\mathbf{E}}\bar{\mathbf{K}}^{\top}, -1)\right]^{-1}, \quad \mathbf{Y} = \mathbf{A}\bar{\mathbf{E}}, \quad \mathbf{U} = \mathbf{A}\mathbf{Z}. \tag{27}$$

Lemma 2 (Unified chunk closed form)**.** *Let* $\mathbf{S}_i = \mathbf{S}_{i-1}\mathbf{D}_i(\mathbf{I} - \boldsymbol{e}_i\boldsymbol{k}_i^{\top}) + \boldsymbol{z}_i\boldsymbol{k}_i^{\top}$ *with* $\mathbf{D}_i = \mathrm{Diag}(\boldsymbol{\alpha}_i)$ *and* $\mathbf{S}_0 = \mathbf{S}^{+}_{[n]}$*. Then*

$$\mathbf{S}_i = \mathbf{S}_0\,\mathrm{Diag}(\boldsymbol{\gamma}_i) + \sum_{j \le i} \boldsymbol{\delta}_j \big(\boldsymbol{k}_j \odot \boldsymbol{\gamma}_i \oslash \boldsymbol{\gamma}_j\big)^{\top}, \qquad \mathbf{\Delta} = \mathbf{U} - \mathbf{Y}\mathbf{S}_0^{\top}, \tag{28}$$

where $\mathbf{\Delta} \in \mathbb{R}^{C \times d_v}$ *stacks the rows* $\boldsymbol{\delta}_j^{\top}$*. Consequently*

$$\mathbf{S}^{-}_{[n+1]} = \mathbf{S}_0\,\mathrm{Diag}(\boldsymbol{\gamma}_C) + \mathbf{\Delta}^{\top}(\mathbf{\Gamma}^{\mathrm{tail}} \odot \mathbf{K}), \qquad \mathbf{O}_{[n]} = \mathbf{Q}_{\gamma}\mathbf{S}_0^{\top} + \mathrm{tril}\big(\mathbf{Q}_{\gamma}\bar{\mathbf{K}}^{\top}\big)\mathbf{\Delta}, \tag{29}$$

with $\mathbf{Q}_{\gamma} = \mathbf{\Gamma}^{\mathrm{pre}} \odot \mathbf{Q}$*.*

Proof. Induct on i. The claim holds at $i = 0$. Assume it at $i - 1$. Since $\mathrm{Diag}(\boldsymbol{\gamma}_{i-1})\mathbf{D}_i = \mathrm{Diag}(\boldsymbol{\gamma}_i)$ and $(\boldsymbol{k}_j \odot \boldsymbol{\gamma}_{i-1} \oslash \boldsymbol{\gamma}_j)^{\top}\mathbf{D}_i = (\boldsymbol{k}_j \odot \boldsymbol{\gamma}_i \oslash \boldsymbol{\gamma}_j)^{\top}$,

$$\mathbf{S}_{i-1}\mathbf{D}_i = \mathbf{S}_0\,\mathrm{Diag}(\boldsymbol{\gamma}_i) + \sum_{j < i} \boldsymbol{\delta}_j \big(\boldsymbol{k}_j \odot \boldsymbol{\gamma}_i \oslash \boldsymbol{\gamma}_j\big)^{\top}.$$

Writing $\boldsymbol{\delta}_i = \boldsymbol{z}_i - \mathbf{S}_{i-1}\mathbf{D}_i\boldsymbol{e}_i$ gives $\mathbf{S}_i = \mathbf{S}_{i-1}\mathbf{D}_i + \boldsymbol{\delta}_i\boldsymbol{k}_i^{\top}$, which is Eq. (28) at i because $\boldsymbol{\gamma}_i \oslash \boldsymbol{\gamma}_i = \mathbf{1}$. Using $\mathrm{Diag}(\boldsymbol{\gamma}_i)\boldsymbol{e}_i = \bar{\mathbf{E}}_{i,:}^{\top}$ and $(\boldsymbol{k}_j \odot \boldsymbol{\gamma}_i \oslash \boldsymbol{\gamma}_j)^{\top}\boldsymbol{e}_i = \bar{\mathbf{K}}_{j,:}\bar{\mathbf{E}}_{i,:}^{\top}$,

$$\boldsymbol{\delta}_i + \sum_{j < i} \big(\bar{\mathbf{E}}\bar{\mathbf{K}}^{\top}\big)_{ij}\boldsymbol{\delta}_j = \boldsymbol{z}_i - \mathbf{S}_0\bar{\mathbf{E}}_{i,:}^{\top},$$

that is $\big[\mathbf{I}_C + \mathrm{tril}(\bar{\mathbf{E}}\bar{\mathbf{K}}^\top, -1)\big]\boldsymbol{\Delta} = \mathbf{Z} - \bar{\mathbf{E}}\mathbf{S}_0^\top$, which is the stated $\boldsymbol{\Delta}$. Evaluating Eq. (28) at $i = C$ gives the state in Eq. (29), and contracting row i with $\boldsymbol{q}_i$ gives the output; the mask includes the diagonal because $j \le i$. □

Specializations. GDN-2 takes $\mathbf{E} = \mathbf{B} \odot \mathbf{K}$ and $\mathbf{Z} = \mathbf{Z}^{\mathrm{write}}$, so Eq. (27) coincides with Eq. (47) and Eq. (29) reproduces Eqs. (49)–(50). KDA takes $\boldsymbol{e}_i = \beta_i \boldsymbol{k}_i$ and $\boldsymbol{z}_i = \beta_i \boldsymbol{v}_i$, so $\bar{\mathbf{E}} = \mathbf{B}_\beta(\boldsymbol{\Gamma}^{\mathrm{pre}} \odot \mathbf{K})$ and $\mathbf{Z} = \mathbf{B}_\beta \mathbf{V}$. Its transform in Eq. (41) equals $\mathbf{A}\mathbf{B}_\beta$ for the $\mathbf{A}$ of Eq. (27), so the lemma's factors coincide with Eq. (42): $\mathbf{Y} = \mathbf{A}\mathbf{B}_\beta(\boldsymbol{\Gamma}^{\mathrm{pre}} \odot \mathbf{K}) = \mathbf{W}$ and $\mathbf{U} = \mathbf{A}\mathbf{B}_\beta\mathbf{V}$. GDN further sets $\boldsymbol{\alpha}_i = \alpha_i \mathbf{1}_{d_k}$ with a scalar α_i, so $\bar{\mathbf{E}}\bar{\mathbf{K}}^\top = \mathrm{Diag}(\boldsymbol{\gamma})\mathbf{B}_\beta\mathbf{K}\mathbf{K}^\top \mathrm{Diag}(\boldsymbol{\gamma})^{-1}$. Diagonal similarity preserves strict lower-triangular structure entrywise, so

$$\mathbf{A} = \mathrm{Diag}(\boldsymbol{\gamma})\big[\mathbf{I}_C + \mathrm{tril}(\mathbf{B}_\beta\mathbf{K}\mathbf{K}^\top, -1)\big]^{-1}\mathrm{Diag}(\boldsymbol{\gamma})^{-1} = \big[\mathbf{I}_C + \mathrm{tril}\big(\mathbf{B}_\beta(\mathbf{K}\mathbf{K}^\top \odot \boldsymbol{\Gamma}), -1\big)\big]^{-1}, \tag{30}$$

whence $\mathbf{Y} = \mathrm{Diag}(\boldsymbol{\gamma})\mathbf{W}$ with $\mathbf{W}$ from Eq. (35), and $\mathbf{U} = \mathbf{A}\mathbf{B}_\beta\mathbf{V}$ coincides with Eq. (36). Equations (37)–(38) then follow from Eq. (29). Numerically, $\bar{\mathbf{K}} = \mathbf{K} \oslash \boldsymbol{\Gamma}^{\mathrm{pre}}$ is never formed explicitly; the kernels evaluate the required ratios through differences of cumulative log-decays, as noted at the end of Appendix F.7.

F.4 Derivation of the Mamba-3 chunk form

For Eq. (53), we claim

$$\mathbf{S}_i = \gamma_i \mathbf{H}_{[n]} + \sum_{j \le i} w_j^{(i)} \mathbf{J}_j, \qquad w_j^{(i)} = \begin{cases} (\gamma_i/\gamma_j)\eta_j, & j < i, \\ c_i, & j = i, \end{cases} \tag{31}$$

with $\mathbf{H}_{[n]}$ and η_j from Eq. (54). At $i = 1$, $\mathbf{S}_1 = \alpha_1(\mathbf{S}_0 + p_1\mathbf{J}_0) + c_1\mathbf{J}_1 = \gamma_1\mathbf{H}_{[n]} + c_1\mathbf{J}_1$. Assume Eq. (31) at $i-1$. Then $\alpha_i\gamma_{i-1} = \gamma_i$; for $j < i-1$, $\alpha_i w_j^{(i-1)} = (\gamma_i/\gamma_j)\eta_j$; for $j = i-1$ the decayed diagonal term $\alpha_i c_{i-1}$ combines with the explicit $\alpha_i p_i \mathbf{J}_{i-1}$ to give $\alpha_i(c_{i-1} + p_i) = (\gamma_i/\gamma_{i-1})\eta_{i-1}$; and $c_i\mathbf{J}_i$ supplies the new diagonal term. This is Eq. (31) at i and matches $\boldsymbol{\Gamma}^{\mathrm{trap}}$ in Eq. (55). Setting $i = C$ gives $w_j^{(C)} = \omega_j$ and $\sum_j \omega_j \boldsymbol{v}_j\boldsymbol{k}_j^\top = \mathbf{V}^\top \mathrm{Diag}(\boldsymbol{\omega})\mathbf{K}$, which is Eq. (56); contracting Eq. (31) with $\boldsymbol{q}_i$ gives Eq. (57). The weight $\eta_C = c_C$ excludes the next token's trapezoidal term, which is why that term is added only after the boundary gate, as in Eq. (59).

F.5 Gated DeltaNet

Cumulative decay and WY factors. For GDN (Yang et al., 2025), define $\gamma_i = \prod_{j=1}^{i} \alpha_j$, $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_C)^\top$, and $\mathbf{B}_\beta = \mathrm{Diag}(\beta_1, \ldots, \beta_C)$. Its causal decay mask is

$$\Gamma_{ij} = \begin{cases} \gamma_i/\gamma_j, & i \ge j, \\ 0, & i < j. \end{cases} \tag{32}$$

Using the original GDN convention, the rows $\boldsymbol{w}_i^\top, \boldsymbol{u}_i^\top$ of the WY factors satisfy

$$\boldsymbol{w}_i = \beta_i \left(\boldsymbol{k}_i - \sum_{j<i} (\boldsymbol{k}_j^\top \boldsymbol{k}_i)\boldsymbol{w}_j \right), \tag{33}$$

$$\boldsymbol{u}_i = \beta_i \left(\boldsymbol{v}_i - \sum_{j<i} \frac{\gamma_i}{\gamma_j} (\boldsymbol{k}_j^\top \boldsymbol{k}_i)\boldsymbol{u}_j \right). \tag{34}$$

The UT form writes these factors as matrix products with unit lower-triangular inverses:

$$\mathbf{W} = \big[\mathbf{I}_C + \mathrm{tril}(\mathbf{B}_\beta\mathbf{K}\mathbf{K}^\top, -1)\big]^{-1}\mathbf{B}_\beta\mathbf{K}, \tag{35}$$

$$\mathbf{U} = \big[\mathbf{I}_C + \mathrm{tril}\big(\mathbf{B}_\beta(\mathbf{K}\mathbf{K}^\top \odot \boldsymbol{\Gamma}), -1\big)\big]^{-1}\mathbf{B}_\beta\mathbf{V}. \tag{36}$$

Here $\mathbf{W} \in \mathbb{R}^{C \times d_k}$ is a WY factor, distinct from the learned $\mathbf{W}_S$. Scalar cumulative decay remains outside $\mathbf{W}$ in the following equations.

Chunkwise computation with SDG. The state and output have the GDN chunkwise form, followed by our boundary update:

$$\mathbf{S}^{-}_{[n+1]} = \gamma_C \mathbf{S}^{+}_{[n]} + \Big(\mathbf{U} - \mathrm{Diag}(\boldsymbol{\gamma})\mathbf{W}(\mathbf{S}^{+}_{[n]})^\top\Big)^\top \mathrm{Diag}(\gamma_C/\boldsymbol{\gamma})\mathbf{K}, \tag{37}$$

$$\mathbf{O}_{[n]} = \mathrm{Diag}(\boldsymbol{\gamma})\mathbf{Q}(\mathbf{S}^{+}_{[n]})^\top + (\mathbf{Q}\mathbf{K}^\top \odot \boldsymbol{\Gamma})\Big(\mathbf{U} - \mathrm{Diag}(\boldsymbol{\gamma})\mathbf{W}(\mathbf{S}^{+}_{[n]})^\top\Big), \tag{38}$$

$$\mathbf{S}^{+}_{[n+1]} = \underbrace{\boldsymbol{\Lambda}_{[n]} \odot \mathbf{S}^{-}_{[n+1]}}_{\text{SDG boundary update}}. \tag{39}$$

The pseudo-value term $\mathbf{U} - \mathrm{Diag}(\boldsymbol{\gamma})\mathbf{W}(\mathbf{S}^{+}_{[n]})^\top$ is shared by the state and output computations. SDG leaves this term and the WY preparation unchanged, and replaces the raw state carried to the next chunk with the final line above.

F.6 Kimi Delta Attention

Channel-wise decay and the UT transform. For KDA (Team et al., 2025), let $\boldsymbol{\gamma}_i = \bigodot_{j=1}^{i} \boldsymbol{\alpha}_j$ and recall Eq. (26),

$$\boldsymbol{\Gamma}^{\mathrm{pre}}_{i,:} = \boldsymbol{\gamma}_i^\top, \qquad \boldsymbol{\Gamma}^{\mathrm{tail}}_{i,:} = (\boldsymbol{\gamma}_C \oslash \boldsymbol{\gamma}_i)^\top. \tag{40}$$

These are the cumulative decay from the chunk start and the remaining decay to its end, respectively. With $\mathbf{B}_\beta = \mathrm{Diag}(\beta_1, \ldots, \beta_C)$, KDA's UT transform and WY factors are

$$\mathbf{A} = \big[\mathbf{I}_C + \mathrm{tril}\big(\mathbf{B}_\beta(\boldsymbol{\Gamma}^{\mathrm{pre}} \odot \mathbf{K})(\mathbf{K} \oslash \boldsymbol{\Gamma}^{\mathrm{pre}})^\top, -1\big)\big]^{-1}\mathbf{B}_\beta, \tag{41}$$

$$\mathbf{W} = \mathbf{A}(\boldsymbol{\Gamma}^{\mathrm{pre}} \odot \mathbf{K}), \qquad \mathbf{U} = \mathbf{A}\mathbf{V}. \tag{42}$$

Here $\mathbf{A}$ denotes the UT transform, reserving $\mathbf{M}_C$ for the causal mask. In this convention, cumulative channel-wise decay is already included in $\mathbf{W}$.

Chunkwise computation with SDG. The resulting state, output, and boundary update are

$$\mathbf{S}^{-}_{[n+1]} = \mathbf{S}^{+}_{[n]}\,\mathrm{Diag}(\boldsymbol{\gamma}_C) + \Big(\mathbf{U} - \mathbf{W}(\mathbf{S}^{+}_{[n]})^\top\Big)^\top (\boldsymbol{\Gamma}^{\mathrm{tail}} \odot \mathbf{K}), \tag{43}$$

$$\mathbf{O}_{[n]} = (\boldsymbol{\Gamma}^{\mathrm{pre}} \odot \mathbf{Q})(\mathbf{S}^{+}_{[n]})^\top + \mathrm{tril}\big((\boldsymbol{\Gamma}^{\mathrm{pre}} \odot \mathbf{Q})(\mathbf{K} \oslash \boldsymbol{\Gamma}^{\mathrm{pre}})^\top\big)\Big(\mathbf{U} - \mathbf{W}(\mathbf{S}^{+}_{[n]})^\top\Big), \tag{44}$$

$$\mathbf{S}^{+}_{[n+1]} = \underbrace{\boldsymbol{\Lambda}_{[n]} \odot \mathbf{S}^{-}_{[n+1]}}_{\text{SDG boundary update}}. \tag{45}$$

The two output terms are the inter-chunk read and intra-chunk masked matrix product. The shared pseudo-values remain $\mathbf{U} - \mathbf{W}(\mathbf{S}^{+}_{[n]})^\top$. KDA's diagonal token decay is retained; SDG adds element-wise decay only after the raw chunk state has been formed.

F.7 Gated DeltaNet-2

Asymmetric erase and write factors. GDN-2 (Hatamizadeh et al., 2026) uses the same $\boldsymbol{\Gamma}^{\mathrm{pre}}$ and $\boldsymbol{\Gamma}^{\mathrm{tail}}$ as KDA. Let $\mathbf{B} \in \mathbb{R}^{C \times d_k}$ and $\mathbf{W}^{\mathrm{write}} \in \mathbb{R}^{C \times d_v}$ stack its erase gates $\boldsymbol{b}_i$ and write gates $\boldsymbol{w}_i$. Following its asymmetric WY form, define

$$\bar{\mathbf{K}} = \mathbf{K} \oslash \boldsymbol{\Gamma}^{\mathrm{pre}}, \qquad \bar{\mathbf{E}} = \boldsymbol{\Gamma}^{\mathrm{pre}} \odot (\mathbf{B} \odot \mathbf{K}), \qquad \mathbf{Z}^{\mathrm{write}} = \mathbf{W}^{\mathrm{write}} \odot \mathbf{V}. \tag{46}$$

The superscript write distinguishes these token-write quantities from the SDG score and WY factors. GDN-2's triangular inverse and erase/write auxiliaries are

$$\mathbf{T} = \mathrm{tril}(\bar{\mathbf{E}}\bar{\mathbf{K}}^\top, -1), \qquad \mathbf{A} = (\mathbf{I}_C + \mathbf{T})^{-1}, \qquad \mathbf{Y} = \mathbf{A}\bar{\mathbf{E}}, \qquad \mathbf{U} = \mathbf{A}\mathbf{Z}^{\mathrm{write}}. \tag{47}$$

The notation $\mathbf{Y}$ follows the erase-side auxiliary in GDN-2. Define its output and tail factors as

$$\mathbf{Q}_\gamma = \boldsymbol{\Gamma}^{\mathrm{pre}} \odot \mathbf{Q}, \qquad \mathbf{K}_{\mathrm{tail}} = \boldsymbol{\Gamma}^{\mathrm{tail}} \odot \mathbf{K}, \qquad \mathbf{A}_{qk} = \mathrm{tril}(\mathbf{Q}_\gamma \bar{\mathbf{K}}^\top). \tag{48}$$

Chunkwise computation with SDG. In the state orientation used here, the GDN-2 equations and our insertion are

$$\mathbf{S}^{-}_{[n+1]} = \mathbf{S}^{+}_{[n]} \operatorname{Diag}(\boldsymbol{\gamma}_C) + \left(\mathbf{U} - \mathbf{Y}(\mathbf{S}^{+}_{[n]})^{\top}\right)^{\top} \mathbf{K}_{\mathrm{tail}}, \tag{49}$$

$$\mathbf{O}_{[n]} = \mathbf{Q}_{\gamma}(\mathbf{S}^{+}_{[n]})^{\top} + \mathbf{A}_{qk}\left(\mathbf{U} - \mathbf{Y}(\mathbf{S}^{+}_{[n]})^{\top}\right), \tag{50}$$

$$\mathbf{S}^{+}_{[n+1]} = \underbrace{\boldsymbol{\Lambda}_{[n]} \odot \mathbf{S}^{-}_{[n+1]}}_{\text{SDG boundary update}}. \tag{51}$$

The erase and write gates enter through $\bar{\mathbf{E}}$ and $\mathbf{Z}^{\mathrm{write}}$, respectively. SDG changes only the matrix carried between chunks; $\mathbf{A}_{qk}$, $\mathbf{K}_{\mathrm{tail}}$, and the native WY preparation are retained. Tying $\boldsymbol{b}_i$ and $\boldsymbol{w}_i$ to β_i recovers KDA, with $\mathbf{Y} = \mathbf{W}$. Under scalar decay, this equals $\operatorname{Diag}(\boldsymbol{\gamma})\mathbf{W}$ in the GDN convention above.

All triangular inverses above denote solves with unit lower-triangular matrices. Decay ratios are evaluated through cumulative log-decays and their differences in the kernels. The mathematical notation does not require explicitly storing inverse cumulative decays.

F.8 MAMBA-3 SISO

Trapezoidal write factors. For Mamba-3 (Lahoti et al., 2026), $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{C \times d_s}$ contain the native normalized, biased, and rotated queries and keys. Let $\delta_i > 0$ be the step size, $\tau_i \in [0, 1]$ the trapezoidal mixing coefficient, and α_i the scalar decay. Define

$$p_i = (1 - \tau_i)\delta_i, \qquad c_i = \tau_i \delta_i, \qquad \mathbf{J}_i = \boldsymbol{v}_i \boldsymbol{k}_i^{\top}. \tag{52}$$

Its native physical-state recurrence, with $\mathbf{S}_0 = \mathbf{S}^{+}_{[n]}$, is

$$\mathbf{S}_i = \alpha_i \mathbf{S}_{i-1} + \alpha_i p_i \mathbf{J}_{i-1} + c_i \mathbf{J}_i, \qquad \boldsymbol{o}_i = \mathbf{S}_i \boldsymbol{q}_i. \tag{53}$$

To give the same inter-chunk plus intra-chunk matrix presentation, set $\gamma_i = \prod_{j=1}^{i} \alpha_j$, $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_C)^{\top}$, and

$$\mathbf{H}_{[n]} = \mathbf{S}^{+}_{[n]} + p_1 \mathbf{J}_0, \qquad \eta_j = \begin{cases} c_j + p_{j+1}, & j < C, \\ c_C, & j = C. \end{cases} \tag{54}$$

Here $\mathbf{J}_0$ uses the preceding token's cached key and value, or is zero at a fresh sequence start; the carry $\mathbf{H}_{[n]}$ is unrelated to the head count H.. The trapezoidal mask and tail weights are

$$\Gamma^{\mathrm{trap}}_{ij} = \begin{cases} (\gamma_i/\gamma_j)\eta_j, & i > j, \\ c_i, & i = j, \\ 0, & i < j, \end{cases} \qquad \omega_j = (\gamma_C/\gamma_j)\eta_j. \tag{55}$$

Chunkwise computation with SDG. The SISO chunk computation can then be written as

$$\mathbf{S}^{-}_{[n+1]} = \gamma_C \mathbf{H}_{[n]} + \mathbf{V}^{\top} \operatorname{Diag}(\boldsymbol{\omega})\mathbf{K}, \tag{56}$$

$$\mathbf{O}_{[n]} = \operatorname{Diag}(\boldsymbol{\gamma})\mathbf{Q}\mathbf{H}^{\top}_{[n]} + (\mathbf{Q}\mathbf{K}^{\top} \odot \boldsymbol{\Gamma}^{\mathrm{trap}})\mathbf{V}, \tag{57}$$

$$\mathbf{S}^{+}_{[n+1]} = \underbrace{\boldsymbol{\Lambda}_{[n]} \odot \mathbf{S}^{-}_{[n+1]}}_{\text{SDG boundary update}}. \tag{58}$$

This has the same separation into an incoming-state read and a causal within-chunk matrix product. Mamba-3 has no delta-rule erase term, so its value factor is $\mathbf{V}$ rather than a WY pseudo-value. Its native skip connection and output gate are applied without modification.

Boundary placement and the next chunk. Equation (25) uses the physical states $\mathbf{S}^{+}_{[n]}$ and $\mathbf{S}^{-}_{[n+1]}$, not the corrected carry $\mathbf{H}_{[n]}$. The last write weight $\eta_C = c_C$ excludes the next token's trapezoidal contribution. Only after SDG has updated the physical state is that contribution added to the next carry:

$$\mathbf{H}_{[n+1]} = \mathbf{S}^{+}_{[n+1]} + p_{[n+1],1} \mathbf{J}_{[n],C}. \tag{59}$$

The saved SDG source is $\mathbf{S}^{+}_{[n+1]}$, before this correction. Thus the gate does not scale the next token's previous-key/value contribution. The rotary phase and previous-key/value caches continue with their native updates.

F.9 REFERENCE BOUNDARY SCAN

The following procedure describes how the state is updated during training, prefill, and decoding. Its cache contains the current state $\mathbf{S}$, the state $\mathbf{S}^{\mathrm{src}}$ at the preceding completed boundary, native auxiliary state $\boldsymbol{\xi}$, and a valid-token counter $r \in \{0, \dots, C-1\}$. The procedure below makes the order of reads, writes, and boundary decay explicit. Let $\mathcal{X}$ denote the projected token inputs and native token gates. For a segment of the sequence, the input and output states satisfy

$$(\mathbf{O}^{\mathrm{seg}}, \mathbf{S}^{\mathrm{out}}, \boldsymbol{\xi}') = \mathcal{F}(\mathcal{X}^{\mathrm{seg}}, \mathbf{S}^{\mathrm{in}}, \boldsymbol{\xi}).$$

At an aligned chunk start, $\mathbf{S}^{\mathrm{in}} = \mathbf{S}^{\mathrm{src}} = \mathbf{S}^{+}_{[n]}$. During continuation of an incomplete chunk, $\mathbf{S}^{\mathrm{in}}$ is instead the current cached state and generally differs from the retained $\mathbf{S}^{\mathrm{src}}$ used by the eventual gate function.

1. **Input.** Native token-local quantities $\mathcal{X}$ for L valid tokens, chunk size C, and cache $(\mathbf{S}, \mathbf{S}^{\mathrm{src}}, \boldsymbol{\xi}, r)$. Set $p \leftarrow 1$ and $\mathbf{O} \leftarrow [\,]$.
2. While $p \le L$, set $m \leftarrow \min\{C - r, L - p + 1\}$ and run the original recurrence on the next contiguous segment.
$$(\mathbf{O}^{\mathrm{seg}}, \mathbf{S}^{\mathrm{out}}, \boldsymbol{\xi}') \leftarrow \mathcal{F}(\mathcal{X}_{p:p+m-1}, \mathbf{S}, \boldsymbol{\xi}).$$
3. Append $\mathbf{O}^{\mathrm{seg}}$ to $\mathbf{O}$ and set $\mathbf{S} \leftarrow \mathbf{S}^{\mathrm{out}}$, $\boldsymbol{\xi} \leftarrow \boldsymbol{\xi}'$, $r \leftarrow r + m$, and $p \leftarrow p + m$.
4. If $r = C$, compute $\boldsymbol{\Lambda} \leftarrow \mathcal{G}_\theta(\mathbf{S}^{\mathrm{src}}, \mathbf{S}^{\mathrm{out}})$ and then set $\mathbf{S} \leftarrow \boldsymbol{\Lambda} \odot \mathbf{S}^{\mathrm{out}}$, $\mathbf{S}^{\mathrm{src}} \leftarrow \mathbf{S}$, and $r \leftarrow 0$. The segment outputs and native auxiliary update have already been formed. The boundary changes neither.
5. Repeat Steps 2–4 and return $\mathbf{O}$, $(\mathbf{S}, \mathbf{S}^{\mathrm{src}}, \boldsymbol{\xi}, r)$.

Proposition 1 (Segmentation invariance)**.** *Assume the base recurrence is segmentation consistent, that is, for every split of a token stream, $\mathcal{F}$ applied to the concatenation equals the sequential composition of $\mathcal{F}$ on the two parts with the intermediate state and auxiliary state passed through. Then for any partition of a stream into consecutive calls, the procedure above returns the same outputs and the same final cache as a single call on the whole stream, and both agree with the chunkwise definition of Eq. (3) with boundaries at absolute positions that are multiples of C.*

Proof. Induct on the number of calls, starting from $r = 0$ at a fresh sequence. Step 2 chooses $m = \min\{C - r, L - p + 1\}$, so no segment crosses a boundary and the hypothesis applies to each segment. The counter r therefore records the absolute position modulo C, and the set of positions at which Step 4 fires is determined by absolute positions alone, independently of the partition. Between two consecutive firings, $\mathbf{S}$ and $\boldsymbol{\xi}$ evolve only through $\mathcal{F}$, whose composition is split invariant by assumption, and $\mathbf{S}^{\mathrm{src}}$ is written only in Step 4. Hence at the boundary after chunk n the two arguments of $\mathcal{G}_\theta$ are exactly $\mathbf{S}^{+}_{[n]}$ and $\mathbf{S}^{-}_{[n+1]}$, as in Eq. (3), and the outputs of each segment are formed before the gate is applied. For Mamba-3 the auxiliary state carries the previous key and value, and the gate acts on the physical state, so Eq. (59) composes in the same way. □

The identity holds in exact arithmetic; in finite precision the two execution paths group the same products differently, so our correctness tests compare them with a tolerance rather than bitwise. Consequently training with the chunkwise form, prefill in one call, and decoding one token per call define the same function of the token stream.

The remaining conventions are implementation choices. For a fresh sequence $\mathbf{S} = \mathbf{S}^{\mathrm{src}}$ is the supplied initial state or zero and $\boldsymbol{\xi}$ uses the base model's initialization; each nonempty packed example owns an independent cache tuple, and empty packed examples are rejected. Masked padding neither invokes the recurrence nor advances r in fresh, prefill, and packed execution, whereas the GDN-2, KDA, and Mamba-3 cached one-token paths assume that every cache row supplies one valid token. In the boundary-offset diagnostic, the inserted filler tokens are valid inputs that enter the recurrence and advance r, thereby shifting answer generation relative to chunk boundaries. A cache is retained only while continuing the same sequence and is reset when it ends. Step 4 also shows why $\mathbf{S}^{\mathrm{src}}$ must not be overwritten while an incomplete tail accumulates.

F.10 Model implementations

GDN, GDN-2, and KDA. All three adapters return current-chunk outputs before applying SDG. Their native token gates, delta residuals, WY constructions, and output gates remain unchanged. The optimized implementations replace or extend only the persistent inter-chunk state scan and its vector–Jacobian product.

Mamba-3. For the bundled SISO recurrence, the native recurrent tuple is

$$\left(\boldsymbol{s}^{\mathrm{angle}}, \mathbf{S}^{\mathrm{ssm}}, \boldsymbol{k}^{\mathrm{prev}}, \boldsymbol{v}^{\mathrm{prev}}\right). \tag{60}$$

SDG additionally stores the state $\mathbf{S}^{\mathrm{src,ssm}}$ and tail counter r. Only $\mathbf{S}^{\mathrm{ssm}}$ is passed to $\mathcal{G}_\theta$ and multiplied by its decay gate. All other tuple components come directly from the native transition.

Supported configurations. The GDN adapter supports key and value head dimensions up to 128. The GDN-2 and KDA adapters currently require equal numbers of value and query/key heads. Their fused paths support $d_k, d_v \le 128$, with wider states using a fallback path. The current projected state-change configurations use boundary recomputation. The Mamba-3 adapter targets the bundled SISO implementation and exposes only successive one-token cached decoding calls. These are implementation limits, not requirements of the method.

G Complete proofs

G.1 Adaptive element-wise ridge

Remark 1 is stated in Section 4.1. The following argument establishes the identity and its converse, followed by the backbone-specific token-update interpretations (Duchi & Singer, 2009; Duchi et al., 2010; Parikh & Boyd, 2014).

Proof of Remark 1. Write $\mathbf{S} = \mathbf{S}^{-}_{[n+1]}$, $\boldsymbol{\rho} = \boldsymbol{\rho}_{[n]}$, and $\boldsymbol{\kappa} = e^{\boldsymbol{\rho}} - \mathbf{1}$. The objective

$$J(\mathbf{M}) = \tfrac{1}{2}\|\mathbf{M} - \mathbf{S}\|_F^2 + \tfrac{1}{2}\sum_{a,b} \kappa_{ab} M_{ab}^2$$

is separable across entries and strictly convex because $1 + \kappa_{ab} = e^{\rho_{ab}} > 0$. From $\partial J/\partial M_{ab} = e^{\rho_{ab}} M_{ab} - S_{ab}$, the unique minimizer is $M_{ab} = e^{-\rho_{ab}} S_{ab} = \Lambda_{ab} S_{ab}$, which is Eq. (7). For the converse, we show that the quadratic is the only separable differentiable penalty with this proximal operator, up to an additive constant. Let $g(\mathbf{M}) = \sum_{a,b} g_{ab}(M_{ab})$ be separable and differentiable with $\mathrm{prox}_g(\mathbf{S}) = \boldsymbol{\Lambda} \odot \mathbf{S}$ for every $\mathbf{S}$. Stationarity at $M_{ab} = \Lambda_{ab} S_{ab}$ reads $\Lambda_{ab} S_{ab} - S_{ab} + g'_{ab}(\Lambda_{ab} S_{ab}) = 0$ for all S_{ab}, hence $g'_{ab}(u) = (\Lambda_{ab}^{-1} - 1)u$ and $g_{ab}(u) = \frac{1}{2}\kappa_{ab} u^2 + \mathrm{const}$ with $\kappa_{ab} = \Lambda_{ab}^{-1} - 1 = e^{\rho_{ab}} - 1$. Boundary decay is thus the prox of a uniquely determined ridge penalty, and $\boldsymbol{\kappa} = \boldsymbol{\rho} + O(\boldsymbol{\rho}^{\odot 2})$ as $\boldsymbol{\rho} \to \mathbf{0}$. □

When the token update is a gradient step. Let $\widetilde{\mathbf{S}} = \mathbf{S}_{t-1}\mathbf{D}_t$ be the state after token decay, so that Eqs. (19)–(22) all take the form

$$\mathbf{S}_t = \widetilde{\mathbf{S}} - \left(\widetilde{\mathbf{S}}\boldsymbol{e}_t - \boldsymbol{z}_t\right)\boldsymbol{k}_t^\top. \tag{61}$$

Lemma 3. *Fix $\boldsymbol{k}_t \neq \mathbf{0}$, $\boldsymbol{e}_t$, and $\boldsymbol{z}_t$. The field $\mathbf{F}(\widetilde{\mathbf{S}}) = (\widetilde{\mathbf{S}}\boldsymbol{e}_t - \boldsymbol{z}_t)\boldsymbol{k}_t^\top$ is the gradient of a scalar function of $\widetilde{\mathbf{S}}$ with respect to the Frobenius inner product if and only if $\boldsymbol{e}_t = c\,\boldsymbol{k}_t$ for some $c \in \mathbb{R}$.*

Proof. $\mathbf{F}$ is affine with constant Jacobian $\mathcal{A}(\mathbf{H}) = (\mathbf{H}\boldsymbol{e}_t)\boldsymbol{k}_t^\top$, and

$$\langle \mathcal{A}(\mathbf{H}), \mathbf{G}\rangle_F = \boldsymbol{e}_t^\top \mathbf{H}^\top \mathbf{G}\boldsymbol{k}_t = \left\langle \mathbf{H}, (\mathbf{G}\boldsymbol{k}_t)\boldsymbol{e}_t^\top \right\rangle_F,$$

so $\mathcal{A}^*(\mathbf{G}) = (\mathbf{G}\boldsymbol{k}_t)\boldsymbol{e}_t^\top$. On a convex domain a C^1 field is a gradient field if and only if its Jacobian is self-adjoint, here if and only if $\mathbf{H}(\boldsymbol{e}_t\boldsymbol{k}_t^\top - \boldsymbol{k}_t\boldsymbol{e}_t^\top) = \mathbf{0}$ for all $\mathbf{H}$, that is $\boldsymbol{e}_t\boldsymbol{k}_t^\top = \boldsymbol{k}_t\boldsymbol{e}_t^\top$. Right-multiplying by $\boldsymbol{k}_t$ gives $\boldsymbol{e}_t = \boldsymbol{k}_t(\boldsymbol{e}_t^\top \boldsymbol{k}_t)/\|\boldsymbol{k}_t\|_2^2$. Conversely $\boldsymbol{e}_t = c\boldsymbol{k}_t$ with $c \neq 0$ gives $\mathbf{F} = \nabla_{\widetilde{\mathbf{S}}}\left[\frac{c}{2}\|\widetilde{\mathbf{S}}\boldsymbol{k}_t - \boldsymbol{z}_t/c\|_2^2\right]$, and $c = 0$ gives a constant field. □

Remark 2 (GDN-2 as a preconditioned gradient step)**.** Let $\boldsymbol{b}_t > \mathbf{0}$, $\tilde{\boldsymbol{k}}_t = \boldsymbol{b}_t \odot \boldsymbol{k}_t$, and $\ell_t(\mathbf{S}) = \frac{1}{2}\|\mathbf{S}\tilde{\boldsymbol{k}}_t - \boldsymbol{z}_t\|_2^2$, so $\nabla\ell_t(\widetilde{\mathbf{S}}) = (\widetilde{\mathbf{S}}\tilde{\boldsymbol{k}}_t - \boldsymbol{z}_t)\tilde{\boldsymbol{k}}_t^\top$. Since $\tilde{\boldsymbol{k}}_t^\top \operatorname{Diag}(\boldsymbol{b}_t)^{-1} = \boldsymbol{k}_t^\top$,

$$\nabla\ell_t(\widetilde{\mathbf{S}}) \operatorname{Diag}(\boldsymbol{b}_t)^{-1} = (\widetilde{\mathbf{S}}\tilde{\boldsymbol{k}}_t - \boldsymbol{z}_t)\boldsymbol{k}_t^\top = \mathbf{F}(\widetilde{\mathbf{S}}), \tag{62}$$

so $\mathbf{F}$ is the gradient of ℓ_t in the key-weighted inner product $\langle \mathbf{A}, \mathbf{B}\rangle = \operatorname{tr}(\mathbf{A}\operatorname{Diag}(\boldsymbol{b}_t)\mathbf{B}^\top)$: GDN-2's token update is a right-preconditioned gradient step. After the chunk's token updates, the boundary decay is the separate proximal step characterized by Remark 1.

DeltaNet, GDN, and KDA have $\boldsymbol{e}_t = \beta_t\boldsymbol{k}_t$ and $\boldsymbol{z}_t = \beta_t\boldsymbol{v}_t$, so Eq. (61) is $\widetilde{\mathbf{S}} - \beta_t\nabla\ell_t(\widetilde{\mathbf{S}})$ with ℓ_t from Eq. (18): an exact gradient step at each token. A chunk applies these local updates sequentially before the boundary proximal step; their composition is not generally one gradient step on a single loss. GDN-2 has $\boldsymbol{e}_t = \boldsymbol{b}_t \odot \boldsymbol{k}_t$ and meets the condition of Lemma 3 only when $b_{t,j}$ is constant on $\{j : k_{t,j} \neq 0\}$. When $\boldsymbol{b}_t > \mathbf{0}$, Remark 2 instead gives a preconditioned-gradient interpretation. If an erase-gate entry is zero, $\operatorname{Diag}(\boldsymbol{b}_t)$ is singular and this positive-definite metric interpretation does not apply. Mamba-3's recurrence contains no read-and-correct term.

G.2 Conditional shrinkage and excess risk

Section 4.2 states Lemma 1 and Corollary 1. This appendix gives their proofs, the readout connection, oracle-rate interpretation, and detailed conditional and projection calculations.

Isotropic readout risk. For the reconstruction risk defined in Section 4.2, let $\widehat{\mathbf{S}} = \boldsymbol{\Lambda} \odot \mathbf{S}^-$ and take $\boldsymbol{q}$ independent of the reconstruction error, with $\mathbb{E}[\boldsymbol{q}\boldsymbol{q}^\top] = d_k^{-1}\mathbf{I}$. Then

$$\mathbb{E}\big[\|(\widehat{\mathbf{S}} - \mathbf{S}^\star)\boldsymbol{q}\|_2^2\big] = \frac{1}{d_k}\,\mathbb{E}\big[\|\widehat{\mathbf{S}} - \mathbf{S}^\star\|_F^2\big]. \tag{63}$$

Other query models can favor different shrinkage; Appendix G.3 compares this criterion with querying a stored key.

Let Φ denote the information available to an oracle gate, and call a gate *admissible* if it is a function of Φ with finite risk. The full boundary-state pair $\Phi = (\mathbf{S}_{[n]}^+, \mathbf{S}_{[n+1]}^-)$ provides one oracle reference. The implemented gate uses only $\Delta\mathbf{S}_{[n]}$ through Section 3.2's parameterization, so it belongs to a restricted subclass. The difference need not contain all the information in the pair.

Oracle rate under conditional uncorrelatedness. Under the additional condition $\mathbb{E}[\mathbf{S}^\star \odot \mathbf{N} \mid \Phi] = \mathbf{0}$ in Lemma 1, the oracle retention on entries with $m_{\star,ab} > 0$ and $m_{N,ab} > 0$ satisfies

$$\boldsymbol{\kappa}^\star = \mathbf{m}_N \oslash \mathbf{m}_\star, \qquad \boldsymbol{\rho}^\star = \log(\mathbf{1} + \mathbf{m}_N \oslash \mathbf{m}_\star)\,. \tag{64}$$

Thus both the optimal ridge weight and decay rate are finite and strictly positive on these entries. If $m_{N,ab} = 0$, the oracle is the identity gate with $\rho_{ab}^\star = \kappa_{ab}^\star = 0$; if $m_{\star,ab} = 0$, it is the zero gate, obtained only as $\rho_{ab}, \kappa_{ab} \to \infty$. Because finite scores yield retention strictly between zero and one, these endpoint gates occur only as limits. This is an oracle identity for moments conditioned on Φ, not a claim that the learned gate explicitly estimates those moments.

Proof of Lemma 1. Fix an entry (a, b) and abbreviate $x = S_{ab}^\star$, $y = S_{ab}^-$, $\lambda = \Lambda_{ab}$, $m = \mathbb{E}[y^2 \mid \Phi] > 0$, and $c = \mathbb{E}[xy \mid \Phi]$. Because λ is Φ-measurable it factors out of the conditional expectation, so

$$\mathbb{E}\big[(\lambda y - x)^2 \mid \Phi\big] = \lambda^2 m - 2\lambda c + \mathbb{E}[x^2 \mid \Phi] = m\Big(\lambda - \frac{c}{m}\Big)^2 + \mathbb{E}[x^2 \mid \Phi] - \frac{c^2}{m}. \tag{65}$$

The conditional minimizer is $\lambda^\star = c/m$, which is Eq. (9) element-wise; summing over entries and taking expectations gives Eq. (10) with $\mathcal{R}(\boldsymbol{\Lambda}^\star) = \mathbb{E}\sum_{a,b}(\mathbb{E}[x^2 \mid \Phi] - c^2/m)$. Under $\mathbb{E}[\mathbf{S}^\star \odot \mathbf{N} \mid \Phi] = \mathbf{0}$ we get $c = \mathbf{m}_\star$ and $m = \mathbf{m}_\star + \mathbf{m}_N$, both non-negative with $m > 0$, so the ratio lies in $[0, 1]$. Conditional Cauchy–Schwarz gives $\mathbb{E}[m(\lambda^\star)^2] = \mathbb{E}[c^2/m] \le \mathbb{E}[x^2] < \infty$, so $\boldsymbol{\Lambda}^\star$ has finite weighted norm. □

The noise-to-signal interpretation in Eq. (64) requires conditional uncorrelatedness for the specified Φ; unconditional orthogonality alone does not imply this condition.

Proof of Corollary 1. Let $\mathcal{H}$ be the space of Φ-measurable $d_v \times d_k$ random matrices with $\mathbb{E}\|\mathbf{m}_-^{1/2} \odot \mathbf{\Lambda}\|_F^2 < \infty$, with inner product $\langle \mathbf{A}, \mathbf{B} \rangle_{\mathbf{m}} = \mathbb{E} \sum_{a,b} m_{-,ab} A_{ab} B_{ab}$ and induced norm $\| \cdot \|_{\mathbf{m}}$. By Lemma 1, $\mathcal{R}(\mathbf{\Lambda}) = \mathcal{R}(\mathbf{\Lambda}^\star) + \|\mathbf{\Lambda} - \mathbf{\Lambda}^\star\|_{\mathbf{m}}^2$, so minimizing $\mathcal{R}$ over a closed linear subspace is orthogonal projection of $\mathbf{\Lambda}^\star$ onto it. The constant families form nested finite-dimensional subspaces $\mathcal{C}_{\text{scalar}} \subset \mathcal{C}_{\text{key}} \subset \mathcal{C}_{\text{elem}} \subset \mathcal{H}$, so their projections compose and

$$\|\mathbf{\Lambda}^\star - \Pi_{\text{scalar}} \mathbf{\Lambda}^\star\|_{\mathbf{m}}^2 = \|\mathbf{\Lambda}^\star - \Pi_{\text{elem}} \mathbf{\Lambda}^\star\|_{\mathbf{m}}^2 + \|\Pi_{\text{elem}} \mathbf{\Lambda}^\star - \Pi_{\text{key}} \mathbf{\Lambda}^\star\|_{\mathbf{m}}^2 + \|\Pi_{\text{key}} \mathbf{\Lambda}^\star - \Pi_{\text{scalar}} \mathbf{\Lambda}^\star\|_{\mathbf{m}}^2, \tag{66}$$

because the first difference is orthogonal to $\mathcal{C}_{\text{elem}}$, which contains the other two, and the second is orthogonal to $\mathcal{C}_{\text{key}}$, which contains the third. For a constant $\mathbf{c}$, $\|\mathbf{\Lambda}^\star - \mathbf{c}\|_{\mathbf{m}}^2 = \sum_{a,b} \mathbb{E}[m_{-,ab}(\Lambda_{ab}^\star - c_{ab})^2]$ is minimized at

$$c_{ab} = \frac{\mathbb{E}[m_{-,ab} \Lambda_{ab}^\star]}{\bar{m}_{ab}} = \frac{\mathbb{E}\big[\mathbb{E}[S_{ab}^\star S_{ab}^- \mid \Phi]\big]}{\bar{m}_{ab}} = \bar{\Lambda}_{ab},$$

using $\mathbf{m}_- \odot \mathbf{\Lambda}^\star = \mathbb{E}[\mathbf{S}^\star \odot \mathbf{S}^- \mid \Phi]$ and the tower property. Restricted to constants the inner product is $\sum_{a,b} \bar{m}_{ab} A_{ab} B_{ab}$, so projecting $\bar{\mathbf{\Lambda}}$ onto row-constant matrices gives μ_b and onto multiples of $\mathbf{1}$ gives μ; the two expressions for μ agree because $\sum_b \bar{m}_{\cdot b} \mu_b = \sum_{a,b} \bar{m}_{ab} \bar{\Lambda}_{ab}$. Substituting into Eq. (66) gives Eq. (11). Each projection is an $\bar{\mathbf{m}}$-weighted average of values of $\mathbf{\Lambda}^\star$, so the box constraint is inactive whenever $\mathbf{\Lambda}^\star \in [0,1]$ almost surely. □

Corollary 2 (A second chain: the cost of adaptive sharing)**.** *Let* $\mathcal{C}_{\text{key}}^{\text{adapt}} = \{\mathbf{1}_{d_v} \boldsymbol{\lambda}(\Phi)^\top : \boldsymbol{\lambda}\ \Phi\text{-measurable}\} \cap \mathcal{H}$. *Then* $\mathcal{C}_{\text{scalar}} \subset \mathcal{C}_{\text{key}} \subset \mathcal{C}_{\text{key}}^{\text{adapt}} \subset \mathcal{C}_{\text{adapt}}$ *is a second chain whose middle projection is*

$$\lambda_b(\Phi) = \frac{\sum_a m_{-,ab} \Lambda_{ab}^\star}{\sum_a m_{-,ab}} = \frac{\sum_a \mathbb{E}[S_{ab}^\star S_{ab}^- \mid \Phi]}{\sum_a \mathbb{E}[(S_{ab}^-)^2 \mid \Phi]},$$

and

$$\min_{\mathcal{C}_{\text{scalar}}} \mathcal{R} - \min_{\mathcal{C}_{\text{adapt}}} \mathcal{R} = \underbrace{\mathbb{E} \sum_{a,b} m_{-,ab} \big(\Lambda_{ab}^\star - \lambda_b(\Phi)\big)^2}_{\mathcal{E}_{\text{value}}^{\text{cond}}} + \underbrace{\mathbb{E} \sum_{a,b} m_{-,ab} \big(\lambda_b(\Phi) - \mu_b\big)^2}_{\mathcal{E}_{\text{state}}^{\text{key}}} + \mathcal{E}_{\text{key}}. \tag{67}$$

Proof. $\mathcal{C}_{\text{key}}^{\text{adapt}}$ is a closed linear subspace of $\mathcal{H}$ containing $\mathcal{C}_{\text{key}}$, so the proof of Corollary 1 applies with $\mathcal{C}_{\text{elem}}$ replaced by $\mathcal{C}_{\text{key}}^{\text{adapt}}$. Minimizing $\mathbb{E} \sum_{a,b} m_{-,ab} (\Lambda_{ab}^\star - \lambda_b)^2$ pointwise in Φ and b gives the stated $\lambda_b(\Phi)$, and the second form follows from $\mathbf{m}_- \odot \mathbf{\Lambda}^\star = \mathbb{E}[\mathbf{S}^\star \odot \mathbf{S}^- \mid \Phi]$. The last term is the same $\mathcal{E}_{\text{key}}$ because both chains project onto $\mathcal{C}_{\text{key}}$ and then onto $\mathcal{C}_{\text{scalar}}$. □

The two chains share $\mathcal{E}_{\text{key}}$ but are not comparable term by term, since $\mathcal{C}_{\text{elem}}$ and $\mathcal{C}_{\text{key}}^{\text{adapt}}$ are not nested: one fixes the gate while keeping element-wise freedom, the other allows adaptation to Φ while sharing across value rows. These families provide idealized counterparts to the variants in Table 4: element-wise SDG belongs to a restricted subclass of $\mathcal{C}_{\text{adapt}}$, channel-wise SDG to a restricted subclass of $\mathcal{C}_{\text{key}}^{\text{adapt}}$, the learned static gate to $\mathcal{C}_{\text{elem}}$, and fixed boundary decay to $\mathcal{C}_{\text{scalar}}$. The excess-risk formulas compare the optima of these families under one fixed state distribution, not the trained variants themselves.

Remark 3 (Bayes optimality under state dependence)**.** Suppose $\mathbf{S}^-$ is Φ-measurable, as for the full boundary-state oracle. Then $\mathbf{m}_- = \mathbf{S}^- \odot \mathbf{S}^-$ and $\mathbf{\Lambda}^\star = \mathbb{E}[\mathbf{S}^\star \mid \Phi] \oslash \mathbf{S}^-$, so

$$\mathbf{\Lambda}^\star \odot \mathbf{S}^- = \mathbb{E}[\mathbf{S}^\star \mid \Phi], \tag{68}$$

the minimum mean-squared-error estimator of $\mathbf{S}^\star$ among *all* Φ-measurable estimators, not only multiplicative ones; $\mathcal{R}(\mathbf{\Lambda}^\star)$ is thus the Bayes risk. Restricting the gate to $(0,1]$ attains Eq. (68) only where $0 < \mathbb{E}[S_{ab}^\star \mid \Phi]/S_{ab}^- \le 1$, that is, where the posterior mean shrinks the observed entry without changing sign, whereas a constant gate is confined to linear shrinkage however informative Φ is. These are conditions for the oracle family. The implemented SDG gate uses only $\Delta \mathbf{S}_{[n]}$, which need not determine $\mathbf{S}^-$, and its finite-score parameterization yields gates strictly between zero and one.

G.3 SUPERPOSITION MEMORY AND TWO SHRINKAGE CRITERIA

Example: superposition memory. Let $\mathbf{S}^\star = \sum_{i\in\mathcal{F}} \boldsymbol{v}_i \boldsymbol{k}_i^\top$ and $\mathbf{N} = \sum_{j\in\mathcal{O}} \boldsymbol{u}_j \boldsymbol{g}_j^\top$ collect useful and interfering associations. Take independent keys uniform on the unit sphere. Values are zero-mean, independent across associations and of the keys, with $\mathbb{E}[v_{i,a}^2] = \sigma_a^2$ and $\mathbb{E}[u_{j,a}^2] = \tau_a^2$. The index sets are independent of both keys and values. For this example, let the oracle observe only the counts, $\Phi_{\mathrm{cnt}} = (|\mathcal{F}|, |\mathcal{O}|)$. Assume $|\mathcal{F}| > 0$ almost surely, $\mathbb{E}|\mathcal{F}| < \infty$, $0 < \mathbb{E}|\mathcal{O}| < \infty$, and $0 < \sigma_a^2, \tau_a^2 < \infty$ for every value channel. These conditions make the following conditional moments and ratios well-defined. The optimal counts-only gate is

$$\Lambda_{ab}^{\star,\mathrm{cnt}} = \frac{|\mathcal{F}|\sigma_a^2}{|\mathcal{F}|\sigma_a^2 + |\mathcal{O}|\tau_a^2} = \Big(1 + \tfrac{|\mathcal{O}|}{|\mathcal{F}|} \cdot \tfrac{\tau_a^2}{\sigma_a^2}\Big)^{-1}. \tag{69}$$

The following calculation derives this gate and the associated risk terms.

Element-wise moments. With $\mathbf{S}^\star = \sum_{i\in\mathcal{F}} \boldsymbol{v}_i \boldsymbol{k}_i^\top$ and $\mathbf{N} = \sum_{j\in\mathcal{O}} \boldsymbol{u}_j \boldsymbol{g}_j^\top$ as defined above, entry (a, b) satisfies $S_{ab}^\star = \sum_{i\in\mathcal{F}} v_{i,a} k_{i,b}$. The values are independent across associations, zero-mean, and independent of the keys, so all cross terms vanish and

$$\mathbb{E}\big[(S_{ab}^\star)^2 \mid \mathcal{F}, \mathcal{O}\big] = \sum_{i\in\mathcal{F}} \mathbb{E}[v_{i,a}^2]\,\mathbb{E}[k_{i,b}^2] = \frac{|\mathcal{F}|\sigma_a^2}{d_k}, \qquad \mathbb{E}\big[N_{ab}^2 \mid \mathcal{F}, \mathcal{O}\big] = \frac{|\mathcal{O}|\tau_a^2}{d_k}, \tag{70}$$

using $\mathbb{E}[\boldsymbol{k}\boldsymbol{k}^\top] = d_k^{-1}\mathbf{I}$ for keys uniform on the unit sphere. The same independence gives $\mathbb{E}[S_{ab}^\star N_{ab} \mid \mathcal{F}, \mathcal{O}] = 0$. Because the index sets are independent of the keys and values, conditioning only on their cardinalities preserves these moments and the zero cross moment. Lemma 1 therefore yields Eq. (69). Both $\Lambda_{ab}^{\star,\mathrm{cnt}}$ and $\bar{m}_{ab}$ are independent of b, so Corollary 1 gives $\mathcal{E}_{\mathrm{key}} = 0$ and

$$\mathcal{E}_{\mathrm{value}} = d_k \sum_a \bar{m}_a \big(\bar{\Lambda}_a - \mu\big)^2, \qquad \bar{m}_a = \frac{\mathbb{E}|\mathcal{F}|\,\sigma_a^2 + \mathbb{E}|\mathcal{O}|\,\tau_a^2}{d_k}, \qquad \bar{\Lambda}_a = \Big(1 + \tfrac{\mathbb{E}|\mathcal{O}|}{\mathbb{E}|\mathcal{F}|} \cdot \tfrac{\tau_a^2}{\sigma_a^2}\Big)^{-1}, \tag{71}$$

Under the stated positivity assumptions, Eq. (71) is strictly positive exactly when τ_a^2/σ_a^2 is not constant across value channels. Moreover, for every channel $(1 + r\tau_a^2/\sigma_a^2)^{-1}$ is strictly decreasing in $r = |\mathcal{O}|/|\mathcal{F}|$. Since the conditional weights are positive, $\mathcal{E}_{\mathrm{state}}(\Phi_{\mathrm{cnt}}) > 0$ exactly when this count ratio is not almost surely constant.

These conclusions apply to the counts-only oracle. Conditioning on the realized state may change both the optimal gate and the conditional signal–noise cross moment, and we assume no information inclusion between Φ_{cnt} and either the boundary-state pair or $\Delta\mathbf{S}_{[n]}$. More generally, if $\sigma(\Phi_1) \subseteq \sigma(\Phi_2)$ under the same state distribution, then

$$\mathcal{E}_{\mathrm{state}}(\Phi_1) \le \mathcal{E}_{\mathrm{state}}(\Phi_2),$$

because the larger admissible family cannot have greater minimum risk, whereas the best constant element-wise risk is unchanged. No such ordering follows without nested information.

Readout-level shrinkage and why the two criteria differ. A complementary calculation places the criterion at the readout rather than the state. Take $m > 1$ independent keys uniform on the unit sphere, values independent, zero-mean, and independent of the keys with $\mathbb{E}[\boldsymbol{v}_j \boldsymbol{v}_j^\top] = \sigma_v^2 \mathbf{I}$, and query with a stored key $\boldsymbol{q} = \boldsymbol{k}_i$:

$$\boldsymbol{y} = \mathbf{S}\boldsymbol{k}_i = \boldsymbol{x} + \boldsymbol{\varepsilon}, \qquad \boldsymbol{x} = \boldsymbol{v}_i, \qquad \boldsymbol{\varepsilon} = \sum_{j\neq i} \boldsymbol{v}_j (\boldsymbol{k}_j^\top \boldsymbol{k}_i). \tag{72}$$

Conditionally on $\boldsymbol{k}_i$, each $z_j = \boldsymbol{k}_j^\top \boldsymbol{k}_i$ has mean zero and second moment $1/d_k$, so

$$\mathbb{E}\|\boldsymbol{x}\|_2^2 = d_v \sigma_v^2, \qquad \mathbb{E}\|\boldsymbol{\varepsilon}\|_2^2 = \frac{m-1}{d_k} d_v \sigma_v^2, \qquad \mathbb{E}[\boldsymbol{x}^\top \boldsymbol{\varepsilon}] = 0, \tag{73}$$

giving $\mathrm{SNR}_{\mathrm{readout}} = d_k/(m-1)$ and, by the same completion of the square, the scalar minimizer

$$\lambda_{\mathrm{readout}}^\star = \frac{d_k}{d_k + m - 1} = \frac{\mathrm{SNR}_{\mathrm{readout}}}{1 + \mathrm{SNR}_{\mathrm{readout}}}. \tag{74}$$

The corresponding state-level quantity for the same memory is different. With one target and $m-1$ interfering associations and $\sigma_a = \tau_a = \sigma_v$, Eq. (70) gives $\Lambda^\star_{ab} = 1/m$ uniformly, whereas Eq. (74) is close to one for $m \ll d_k$. The discrepancy is entirely a difference of query model, not an inconsistency: Eq. (63) weights all d_k directions equally, while Eq. (72) uses a query aligned with the target's key and therefore amplifies the target by a factor d_k relative to the isotropic average. We report the state-level criterion in the main text because it is separable across entries, which is what makes element-wise gate families directly comparable; a general query covariance couples entries and destroys this separability. Neither calculation characterizes the capacity of learned linear attention.

Isotropic future queries. For Eq. (63), condition on $\mathbf{A} = \widehat{\mathbf{S}} - \mathbf{S}^\star$. Independence and isotropy of $\boldsymbol{q}$ give

$$\mathbb{E}_{\boldsymbol{q}}\|\mathbf{A}\boldsymbol{q}\|_2^2 = \operatorname{tr}\big(\mathbf{A}^\top \mathbf{A}\, \mathbb{E}[\boldsymbol{q}\boldsymbol{q}^\top]\big) = \frac{1}{d_k}\|\mathbf{A}\|_F^2. \tag{75}$$

Taking the remaining expectation proves the identity.

H State propagation and value-row structure

Token-wise gates and shared transitions. A direct extension of GDN or KDA replaces the token-wise scalar or diagonal decay with a full matrix gate:

$$\mathbf{S}_t = (\mathbf{\Lambda}_t \odot \mathbf{S}_{t-1})(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top) + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^\top, \qquad \mathbf{\Lambda}_t \in (0,1]^{d_v \times d_k}. \tag{76}$$

This recurrence is well-defined, but its chunkwise computation differs from that of the base layer. In KDA, every value row shares the same key-space transition. At position r within a chunk, its homogeneous transition is

$$\mathbf{P}_r = \prod_{i=1}^{r} \mathbf{D}_i(\mathbf{I} - \beta_i \boldsymbol{k}_i \boldsymbol{k}_i^\top), \tag{77}$$

with earlier factors on the left. The chunk index is suppressed; $\mathbf{P}_C$ is $\mathbf{P}_{[n]}$ in Eq. (2). One set of WY factors therefore serves all value channels (Yang et al., 2025; Team et al., 2025).

For a general element-wise gate, let $\boldsymbol{\lambda}_i^{(a)}$ denote row a of $\mathbf{\Lambda}_i$. Given the realized token gates, row a has homogeneous transition

$$\mathbf{P}_r^{(a)} = \prod_{i=1}^{r} \operatorname{Diag}(\boldsymbol{\lambda}_i^{(a)})(\mathbf{I} - \beta_i \boldsymbol{k}_i \boldsymbol{k}_i^\top). \tag{78}$$

These transitions generally differ across value rows, requiring up to d_v sets of factors instead of the shared set used by existing kernels. If the gates also read the running state, the factors cannot be prepared from token inputs alone. Equation (78) describes propagation along the realized gate path, not the Jacobian of the full state-dependent recurrence. Boundary placement retains the native shared transitions within each chunk.

Boundary-gated state propagation. This section unrolls the boundary-gated recurrence into a temporal kernel over past writes and shows that this kernel is diagonal: it scales each value coordinate separately rather than mixing them.

$$\mathbf{S}^-_{[n+1]} = \mathbf{S}^+_{[n]} \mathbf{P}_{[n]} + \mathbf{\Omega}_{[n]}, \tag{79}$$

where $\mathbf{P}_{[n]} \in \mathbb{R}^{d_k \times d_k}$ summarizes the within-chunk transition and $\mathbf{\Omega}_{[n]}$ collects writes. Vectorize the state column-wise and let

$$\mathbf{D}_{[n]} = \operatorname{Diag}(\operatorname{vec} \mathbf{\Lambda}_{[n]}), \qquad \mathbf{T}_{[n]} = \mathbf{D}_{[n]}(\mathbf{P}_{[n]}^\top \otimes \mathbf{I}_{d_v}). \tag{80}$$

Conditional on the realized gate trajectory,

$$\operatorname{vec}(\mathbf{S}^+_{[n+1]}) = \mathbf{T}_{[n]} \operatorname{vec}(\mathbf{S}^+_{[n]}) + \mathbf{D}_{[n]} \operatorname{vec}(\mathbf{\Omega}_{[n]}), \tag{81}$$

and unrolling gives

$$\operatorname{vec}(\mathbf{S}^+_{[n+1]}) = \left(\prod_{\ell=1}^{n} \mathbf{T}_{[\ell]}\right) \operatorname{vec}(\mathbf{S}^+_{[1]}) + \sum_{r=1}^{n} \left(\prod_{\ell=r+1}^{n} \mathbf{T}_{[\ell]}\right) \mathbf{D}_{[r]} \operatorname{vec}(\mathbf{\Omega}_{[r]}), \tag{82}$$

with later transitions ordered on the left. With the readout $\boldsymbol{o}_t = (\boldsymbol{q}_t^\top \otimes \mathbf{I}_{d_v})\,\mathrm{vec}(\mathbf{S}_t)$, a zero initial state, and the realized transition and gate factors held fixed, Eq. (82) expresses the output as a temporal kernel applied to past writes,

$$\boldsymbol{o}_t = \sum_{j \le t} \boldsymbol{\mathcal{K}}_{t,j} \boldsymbol{z}_j, \qquad \boldsymbol{\mathcal{K}}_{t,j} \in \mathbb{R}^{d_v \times d_v}, \tag{83}$$

where $\boldsymbol{z}_j$ is the write vector of token j: $\boldsymbol{w}_j \odot \boldsymbol{v}_j$ for GDN-2, $\beta_j \boldsymbol{v}_j$ for GDN and KDA, and a trapezoidal combination of $\boldsymbol{v}_{j-1}$ and $\boldsymbol{v}_j$ for Mamba-3. This kernel is not an arbitrary operator.

Lemma 4 (Value-row decoupling)**.** *Suppose the base recurrence has the form*

$$\mathbf{S}_t = \mathbf{S}_{t-1}\mathbf{A}_t + \sum_{\ell=1}^{L} \boldsymbol{z}_t^{(\ell)} \big(\boldsymbol{k}_t^{(\ell)}\big)^\top,$$

with $\mathbf{A}_t \in \mathbb{R}^{d_k \times d_k}$ and all $\boldsymbol{k}_t^{(\ell)}$ determined by token-local quantities. This covers Eqs. (19)–(22) with $L = 1$ and $\mathbf{A}_t = \mathbf{D}_t(\mathbf{I} - \boldsymbol{e}_t \boldsymbol{k}_t^\top)$, and Eq. (53) with $L = 2$ and $\mathbf{A}_t = \alpha_t \mathbf{I}$. Add the boundary update of Eq. (25). Then for every a, row a of the state follows a recurrence involving only row a of the initial state, the shared key-side factors $\{\mathbf{A}_t, \boldsymbol{k}_t^{(\ell)}\}$, the scalar sources $\{z_{t,a}^{(\ell)}\}$, and row a of $\mathbf{G}$. Consequently the kernel of Eq. (83) is diagonal,

$$\boldsymbol{\mathcal{K}}_{t,j} = \mathrm{Diag}\big(c_{t,j}^{(1)}, \ldots, c_{t,j}^{(d_v)}\big). \tag{84}$$

Proof. Row a of the update is $\boldsymbol{s}_{t-1,a}^\top \mathbf{A}_t + \sum_\ell z_{t,a}^{(\ell)} (\boldsymbol{k}_t^{(\ell)})^\top$, because right multiplication acts on the key index only and each outer product contributes to row a alone. At a boundary, row a is multiplied element-wise by row a of $\boldsymbol{\Lambda}_{[n]}$, and row a of $\mathbf{Z}_{[n]} = \mathbf{G} \odot (\Delta\mathbf{S}_{[n]} \mathbf{W}_S^\top)$ equals $\boldsymbol{g}_a \odot (\mathbf{W}_S \Delta \boldsymbol{s}_a)$, which depends only on row a of $\Delta\mathbf{S}_{[n]}$. Both steps preserve the property, so by induction over tokens and boundaries row a never depends on row $a' \neq a$. Since $\boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t$, coordinate a of $\boldsymbol{o}_t$ is a linear functional of row a alone, which is Eq. (84). □

In the base models $c_{t,j}^{(a)}$ does not depend on a, so the kernel is $c_{t,j}\mathbf{I}_{d_v}$: one weight shared by all value coordinates. An element-wise boundary gate makes it a diagonal operator with value-channel-specific entries. Two scopes differ here. Diagonality is *structural* and holds for every realization, because row a of the gate reads only row a of the state change. The *linear* representation in Eq. (83) is conditional on the realized gate path: $\mathbf{D}_{[n]}$ depends on the evolving state, so the coefficients are themselves functions of past values and cannot be precomputed from positions or reduced to one input-independent attention matrix.

Lemma 4 also justifies two implementation choices. Within a chunk $\mathbf{A}_t$ is shared across value rows, since SDG acts only at boundaries, so one set of WY factors still serves all value channels—the requirement identified in Eqs. (77)–(78). Across value rows the augmented recurrence is embarrassingly parallel, so the state scan may be tiled along d_v with no cross-tile communication, while each tile holds full rows along d_k because $\mathbf{W}_S$ mixes key channels.

I KERNEL AND BACKWARD DETAILS

Boundary gate and implementation. For a fixed base rate $\rho_0 > 0$, define

$$\mathrm{squareplus}(z; \rho_0^2) = \tfrac{1}{2}\big(z + \sqrt{z^2 + \rho_0^2}\big), \qquad \rho(z) = 2\,\mathrm{squareplus}(z; \rho_0^2).$$

This is the Squareplus parameterization (Barron, 2021) in Eq. (5). The rate is smooth and increasing, with value ρ_0 and derivative one at zero. If the state is unchanged, then $\Delta\mathbf{S}_{[n]} = \mathbf{0}$ and the gate is $e^{-\rho_0}\mathbf{1}$.

Forward pass. We implement SDG in Triton (Tillet et al., 2019), building on Flash Linear Attention (Yang & Zhang, 2024). The boundary update is fused into the inter-chunk state scan; native factor preparation and output kernels are retained. On supported fused paths, each program holds a state tile in SRAM or registers, forms the difference in FP32, and projects it using Tensor Cores. The projected difference, score, rate, and gate remain on chip, avoiding HBM writes for these forward intermediates. Appendix F gives backbone-specific details and supported configurations.

Backward pass. The backward pass traverses the state scan in reverse order (Hatamizadeh et al., 2026). Differentiating the gate produces gradients for both the raw state and the incoming state. The former enters the native chunk backward computation; the latter is added directly to its incoming-state gradient. We save or recompute boundary states, recompute gate intermediates on chip, and accumulate the gain gradient during the scan. A separate reduction computes the projection gradient from saved gradients of $\mathbf{X}_{[n]}$. The following subsections give the gate VJP and additive reference forward and backward algorithms.

I.1 Gate-function derivatives

For one boundary, write $\mathbf{S}^{\mathrm{src}}$ for the state at the preceding boundary and $\mathbf{S}^{\mathrm{out}}$ for the gated result, and define

$$\begin{aligned} \Delta\mathbf{S} &= \mathbf{S}^{-} - \mathbf{S}^{\mathrm{src}}, \qquad \mathbf{X} = \Delta\mathbf{S}\mathbf{W}_S^{\top}, \\ \mathbf{Z} &= \mathbf{G} \odot \mathbf{X}, \qquad \boldsymbol{\rho} = \mathbf{Z} + \sqrt{\mathbf{Z} \odot \mathbf{Z} + \rho_0^2 \mathbf{1}}, \\ \boldsymbol{\Lambda} &= \exp(-\boldsymbol{\rho}), \qquad \mathbf{S}^{\mathrm{out}} = \boldsymbol{\Lambda} \odot \mathbf{S}^{-}. \end{aligned} \tag{85}$$

For a tensor $\mathbf{T}$, write $\mathrm{d}\mathbf{T} = \partial\mathcal{L}/\partial\mathbf{T}$. Given $\mathrm{d}\mathbf{S}^{\mathrm{out}}$, the gate vector–Jacobian product computes

$$\mathrm{d}\boldsymbol{\rho} = -\mathrm{d}\mathbf{S}^{\mathrm{out}} \odot \mathbf{S}^{\mathrm{out}}, \qquad \mathrm{d}\mathbf{Z} = \mathrm{d}\boldsymbol{\rho} \odot \frac{\boldsymbol{\rho}}{\sqrt{\mathbf{Z} \odot \mathbf{Z} + \rho_0^2 \mathbf{1}}}, \qquad \mathrm{d}\mathbf{X} = \mathrm{d}\mathbf{Z} \odot \mathbf{G}, \tag{86}$$

with element-wise division, and then

$$\mathrm{d}\mathbf{G} = \mathrm{d}\mathbf{Z} \odot \mathbf{X}, \qquad \mathrm{d}\mathbf{W}_S = \mathrm{d}\mathbf{X}^{\top}\Delta\mathbf{S}, \tag{87}$$

summed over chunks and batch examples. The state gradients are

$$\mathrm{d}\mathbf{S}^{-} = \mathrm{d}\mathbf{S}^{\mathrm{out}} \odot \boldsymbol{\Lambda} + \mathrm{d}\mathbf{X}\mathbf{W}_S, \tag{88}$$

$$\mathrm{d}\mathbf{S}^{\mathrm{src}} = -\mathrm{d}\mathbf{X}\mathbf{W}_S, \tag{89}$$

the two gate-path terms having opposite signs because of the subtraction in $\Delta\mathbf{S}$, and $\mathrm{d}\mathbf{S}^{\mathrm{src}}$ is added to the gradient returned by the base chunk recurrence. This gives a consistency check used in our tests: for standard linear attention $\Delta\mathbf{S}_{[n]} = \mathbf{V}_{[n]}^{\top}\mathbf{K}_{[n]}$ is independent of the incoming state, so the two terms must cancel exactly once accumulated into $\mathrm{d}\mathbf{S}_{[n]}^{+}$.

For stable evaluation, the kernel computes $\sqrt{z^2 + \rho_0^2}$ with a scaled norm. When $z < 0$, it evaluates the equivalent expression $\rho_0^2/(\sqrt{z^2 + \rho_0^2} - z)$ to avoid cancellation. The derivative is $\rho(z)/\sqrt{z^2 + \rho_0^2}$, where $\rho(z) = z + \sqrt{z^2 + \rho_0^2}$. This expression is positive on both sides of zero and equals one at zero.

I.2 Additive reference forward pass

Algorithm 2 expands Algorithm 1 for unnormalized additive linear attention. Its on-chip steps illustrate the arithmetic rather than a common kernel launch schedule for all backbones. In this special case, $\Delta\mathbf{S}_{[n]} = \mathbf{V}_{[n]}^{\top}\mathbf{K}_{[n]}$ is independent of the incoming state for fixed chunk inputs. The gated recurrent layers evaluated here retain their native state transitions, so their state differences also depend on the incoming memory. This example assumes complete chunks, $L = NC$; arbitrary lengths and cached continuations follow Appendix F.9, where incomplete final chunks do not trigger boundary decay.

Algorithm 2 SDG: Illustrative Forward Pass for Additive Linear Attention

Require: $L = NC$ and $N = L/C$ complete chunks; $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{L \times d_k}$, $\mathbf{V} \in \mathbb{R}^{L \times d_v}$, initial state $\mathbf{S}^+_{[1]} \in \mathbb{R}^{d_v \times d_k}$, chunk size C, and gate parameters $\mathbf{W}_S, \mathbf{G}$ with fixed base rate ρ_0.

1: Partition $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ into N complete chunks of C tokens, $\{\mathbf{Q}_{[n]}, \mathbf{K}_{[n]}, \mathbf{V}_{[n]}\}_{n=1}^{N}$.
2: Load $\mathbf{S}^+_{[1]}$ from HBM to SRAM and construct the causal mask $\mathbf{M}_C$ on chip.
3: **Load $\mathbf{W}_S, \mathbf{G}$ from HBM to SRAM.**
4: **for** $n = 1, \dots, N$ **do**
5: Load $\mathbf{Q}_{[n]}, \mathbf{K}_{[n]}, \mathbf{V}_{[n]}$ from HBM to SRAM.
6: On chip, compute $\mathbf{O}_{[n]} = \mathbf{Q}_{[n]}(\mathbf{S}^+_{[n]})^\top + (\mathbf{Q}_{[n]}\mathbf{K}^\top_{[n]} \odot \mathbf{M}_C)\mathbf{V}_{[n]}$.
7: On chip, compute $\mathbf{S}^-_{[n+1]} = \mathbf{S}^+_{[n]} + \mathbf{V}^\top_{[n]}\mathbf{K}_{[n]}$.
8: **On chip, compute $\Delta\mathbf{S}_{[n]} = \mathbf{S}^-_{[n+1]} - \mathbf{S}^+_{[n]}$.**
9: **On chip, compute $\mathbf{X}_{[n]} = \Delta\mathbf{S}_{[n]}\mathbf{W}_S^\top, \quad \mathbf{Z}_{[n]} = \mathbf{G} \odot \mathbf{X}_{[n]}$.**
10: **On chip, compute $\mathbf{\Lambda}_{[n]} = \exp\big[-2\,\mathrm{squareplus}(\mathbf{Z}_{[n]}; \rho_0^2)\big]$.**
11: **On chip, compute $\mathbf{S}^+_{[n+1]} = \mathbf{\Lambda}_{[n]} \odot \mathbf{S}^-_{[n+1]}$.**
12: Store $\mathbf{O}_{[n]}$ from SRAM to HBM.
13: **end for**
14: Store $\mathbf{S}^+_{[N+1]}$ from SRAM to HBM.
15: **return** $\mathbf{O} = \mathrm{Concat}(\mathbf{O}_{[1]}, \dots, \mathbf{O}_{[N]})$ and $\mathbf{S}^+_{[N+1]}$.

I.3 Additive reference backward pass

Algorithm 3 gives the reverse chunk scan for the additive example in Algorithm 2. The prefix d denotes the gradient of the loss with respect to a tensor. The quantities $\mathrm{d}\mathbf{S}^-_{[n+1]}$ and $\mathrm{d}\mathbf{S}^{\mathrm{src}}_{[n]}$ are the gradients returned by the boundary update with respect to $\mathbf{S}^-_{[n+1]}$ and $\mathbf{S}^+_{[n]}$. They follow Eqs. (86)–(89). The scan saves $\mathrm{d}\mathbf{X}_{[n]}$ for a separate tiled reduction of the projection gradient.

Algorithm 3 SDG: Fused Backward Pass for Standard Linear Attention

Require: $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, output gradients $\mathrm{d}\mathbf{O}$, saved or recomputed boundary states, and $\mathrm{d}\mathbf{S}^+_{[N+1]}$, with $\mathbf{W}_S, \mathbf{G}, \rho_0$.

1: Initialize parameter gradients to zero and construct $\mathbf{M}_C$ on chip.
2: **Load $\mathbf{W}_S, \mathbf{G}$ from HBM to SRAM.**
3: **for** $n = N, \dots, 1$ **do**
4: Load $\mathbf{Q}_{[n]}, \mathbf{K}_{[n]}, \mathbf{V}_{[n]}, \mathrm{d}\mathbf{O}_{[n]}, \mathbf{S}^+_{[n]}$, and $\mathbf{S}^-_{[n+1]}$ from HBM to SRAM.
5: **On chip, compute $(\mathrm{d}\mathbf{S}^-_{[n+1]}, \mathrm{d}\mathbf{S}^{\mathrm{src}}_{[n]}) = \mathrm{VJP}_{\text{Eqs. (4)–(6)}}(\mathrm{d}\mathbf{S}^+_{[n+1]})$.**
6: **On chip, accumulate $\mathrm{d}\mathbf{G} \mathrel{+}= \mathrm{d}\mathbf{Z}_{[n]} \odot \mathbf{X}_{[n]}$.**
7: **Store $\mathrm{d}\mathbf{X}_{[n]}$ from SRAM to HBM.**
8: On chip, compute $\mathbf{A}_{qk,[n]} = \mathbf{Q}_{[n]}\mathbf{K}^\top_{[n]} \odot \mathbf{M}_C$ and $\mathrm{d}\mathbf{A}_{qk,[n]} = \mathrm{d}\mathbf{O}_{[n]}\mathbf{V}^\top_{[n]}$.
9: On chip, compute $\mathrm{d}\mathbf{Q}_{[n]} = \mathrm{d}\mathbf{O}_{[n]}\mathbf{S}^+_{[n]} + (\mathrm{d}\mathbf{A}_{qk,[n]} \odot \mathbf{M}_C)\mathbf{K}_{[n]}$.
10: On chip, compute $\mathrm{d}\mathbf{K}_{[n]} = (\mathrm{d}\mathbf{A}_{qk,[n]} \odot \mathbf{M}_C)^\top\mathbf{Q}_{[n]} + \mathbf{V}_{[n]}\mathrm{d}\mathbf{S}^-_{[n+1]}$.
11: On chip, compute $\mathrm{d}\mathbf{V}_{[n]} = \mathbf{A}^\top_{qk,[n]}\mathrm{d}\mathbf{O}_{[n]} + \mathbf{K}_{[n]}(\mathrm{d}\mathbf{S}^-_{[n+1]})^\top$.
12: On chip, accumulate $\mathrm{d}\mathbf{S}^+_{[n]} = \mathrm{d}\mathbf{S}^-_{[n+1]} + \mathrm{d}\mathbf{O}^\top_{[n]}\mathbf{Q}_{[n]} + \mathrm{d}\mathbf{S}^{\mathrm{src}}_{[n]}$.
13: Store $\mathrm{d}\mathbf{Q}_{[n]}$, $\mathrm{d}\mathbf{K}_{[n]}$, and $\mathrm{d}\mathbf{V}_{[n]}$ to HBM.
14: **end for**
15: **Load tiles of $\mathrm{d}\mathbf{X}_{[n]}$, $\mathbf{S}^+_{[n]}$, and $\mathbf{S}^-_{[n+1]}$ from HBM to SRAM.**
16: **On chip, compute $\mathrm{d}\mathbf{W}_S = \sum_{n=1}^{N} \mathrm{d}\mathbf{X}^\top_{[n]}(\mathbf{S}^-_{[n+1]} - \mathbf{S}^+_{[n]})$ by tiled reduction.**
17: **return** input, initial-state, and gate-parameter gradients.

I.4 Fused kernel implementation

GDN. The adapter retains GDN's native projections, token gates, WY preparation, intra-chunk interaction, and output path. The fused state scan applies the boundary gate after the raw state of a complete chunk has been formed. Its backward pass either reads saved boundary states or recomputes them according to the configured path.

GDN-2. The native channel-wise WY preparation, intra-chunk interaction, and output kernel are retained. A persistent state program consumes the native transition factors, holds one key/value state tile, computes the boundary branch after a complete chunk, and writes the gated state. Its custom backward mirrors the state scan and can use saved boundary states or recomputation according to the configured path.

KDA. The adapter retains KDA's gate, intra-chunk, output, and local backward kernels. It reuses the boundary state scan where the WY state interface matches GDN-2, while preserving the KDA-specific token transition.

Mamba-3. The boundary branch is integrated into the native SSM matrix-state scan and corresponding backward pass. Rotary phase, trapezoidal quantities, output gates, and auxiliary state recurrences remain unchanged. The one-token decode path advances a device-resident counter. Inactive rows perform no matrix-sized boundary work, while rows completing a C-token interval update the matrix and source cache.

I.5 Complexity and saved state

For sequence lengths L_n, the exact number of newly completed boundaries depends on incoming tail counters r_n and is

$$N_{\mathrm{bnd}} = \sum_n \left\lfloor \frac{r_n + L_n}{C} \right\rfloor . \tag{90}$$

For linear attention layers, forward gate-function work is $\Theta(N_{\mathrm{bnd}} H d_v d_k^2)$ for the projection and $\Theta(N_{\mathrm{bnd}} H d_v d_k)$ for the element-wise operations. For Mamba-3, replace d_k by d_s. A decode cache adds one source matrix and one integer counter per enabled layer and sequence. Training stores or recomputes boundary states. The backward scan also writes $\mathrm{d}\mathbf{X}_{[n]}$ to HBM for the projection-gradient reduction. This reduction has the same asymptotic arithmetic cost as the forward projection.

J FLOPs analysis

Counting convention. We count a multiply–add as two floating-point operations (FLOPs), so a dense $(m \times k)(k \times n)$ product costs $2mkn$ FLOPs. Consider a sequence of length L, H recurrent heads, state dimensions $d_v \times d_k$, and chunk size C. The counts below describe the additional gate computation in one SDG-enabled layer; they exclude the unchanged base recurrence, input/output projections, and feed-forward network. We report the leading matrix-multiplication cost explicitly and group element-wise arithmetic into lower-order terms. Square roots and exponentials are evaluated once per state element in the forward gate; their hardware-dependent costs are not equated to those of an addition or multiplication.

Forward computation. At each completed boundary, the product $\Delta\mathbf{S}\mathbf{W}_S^\top$ costs $2Hd_v d_k^2$ FLOPs. Forming the state difference, applying the gain, evaluating the rate and gate, and scaling the state require $O(Hd_v d_k)$ additional element-wise work. For an aligned sequence with $C \mid L$, there are L/C boundaries, giving

$$F_{\mathrm{SDG,fwd}} = \frac{2LHd_v d_k^2}{C} + O\!\left(\frac{LHd_v d_k}{C}\right), \qquad \frac{F_{\mathrm{SDG,fwd}}}{L} = \frac{2Hd_v d_k^2}{C} + O\!\left(\frac{Hd_v d_k}{C}\right). \tag{91}$$

For arbitrary lengths or cached continuations, replace L/C by the number of completed boundaries: $\lfloor L/C \rfloor$ for a fresh sequence, or N_{bnd} from Eq. (90) for a batch with incoming tail counters. A partial final chunk does not trigger a boundary gate. Ordinary decoding steps perform no gate matrix multiplication; a step completing a boundary pays the full $2Hd_v d_k^2$ projection cost. Thus, amortized cost and individual boundary-step cost differ by a factor of C for complete intervals. For the Mamba-3 matrix state, replace d_k by d_s.

Backward computation and recomputation. The gate derivatives require two matrix products: $\mathrm{d}\mathbf{X}\mathbf{W}_S$ for the state gradients and $\mathrm{d}\mathbf{X}^\top\Delta\mathbf{S}$ for the projection gradient. Each costs $2Hd_v d_k^2$ FLOPs per boundary. The first product is shared by both state-gradient paths in Eqs. (88) and (89); it need

not be evaluated twice. With saved gate intermediates, a forward and backward evaluation therefore costs $6Hd_v d_k^2$ projection FLOPs per boundary. Recomputing $\mathbf{X}$ once during backward, as in the fused gate path, adds another $2Hd_v d_k^2$. Element-wise derivatives and gain-gradient accumulation add lower-order work. Replaying the base recurrence to recover boundary states, further activation checkpointing, and the optimizer update are separate costs and are excluded from these projection counts.

Batch and model scaling. For a batch of B sequences of length L and M SDG-enabled layers with the same dimensions and chunk size, the leading added forward cost is $2BMLHd_v d_k^2/C$. The corresponding forward-plus-backward projection cost is $6BMLHd_v d_k^2/C$ with saved intermediates, or $8BMLHd_v d_k^2/C$ with one gate recomputation. For heterogeneous layers, sum each layer's count instead of multiplying by M. If $d_k = d_v = d$, the forward cost simplifies to

$$F_{\mathrm{SDG,fwd}}^{\mathrm{model}} = \frac{2BMLHd^3}{C} + O\left(\frac{BMLHd^2}{C}\right). \tag{92}$$

Only layers using SDG contribute to this additional cost, including in hybrid models.

Scaling and measured efficiency. For fixed head dimensions and C, the additional FLOPs grow linearly with the number of tokens, and their amortized value decreases as $1/C$. This does not make C a cost-only tuning parameter: changing it also changes when the model applies decay. The relative arithmetic overhead is $F_{\mathrm{SDG}}/F_{\mathrm{base}}$ under the same counting scope, and cannot be inferred from the added parameter count alone because gate parameters are used once per boundary. Nor does a FLOP ratio predict a latency ratio. The gate runs in the sequential state scan; matrix size, Tensor Core utilization, register pressure, and memory traffic affect its runtime. Fusion avoids materializing forward gate intermediates in HBM, whereas backward recomputation and the projection-gradient reduction introduce additional work and traffic. The end-to-end throughput and latency measurements in Section 5 complement these arithmetic counts.