HISTORY REPLAYS ITSELF: UNLEASHING LONG-CONTEXT CAPABILITY IN LINEAR ATTENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Linear attention promises the efficiency needed for long-context language modelling, yet its fixed-size recurrent state must simultaneously support ongoing computation and preserve information from an ever-growing history. This coupling has reinforced a prevailing view that the state's memory capacity defines the model's long-context capability. We challenge this presumed ceiling: *recurrent computation need not bear the full burden of long-term storage*. We introduce NoteBank, a sparse episodic memory that archives selected text before future queries are known, uses the backbone's native features for writing and addressing, and reads retrieved evidence through its native forward computation via replay. This *training-free, plug-and-play* interface requires no external embedding model, separately trained retriever, or separate neural reader. At one million tokens, it raises Gated DeltaNet's mean accuracy across six RULER retrieval tasks from 1.7% to 70.3%, with no observed significant retrieval degradation from 4K to 1M tokens. This recovery adds $\leqslant$0.025% to analytical FLOPs, with persistent context storage approximately 1/2600 that of a Transformer's full MHA KV cache at matched width, depth, and precision. Experiments extend from three 1.3B recurrent backbones to Mamba-2-8B and post-trained Qwen3.5-9B, demonstrating gains on LongBench v1/v2 and MemoryAgentBench. Together, these results show that sparse evidence storage and native replay can substantially expand the long-context capability of pretrained linear attention models while retaining the efficiency of recurrent computation.

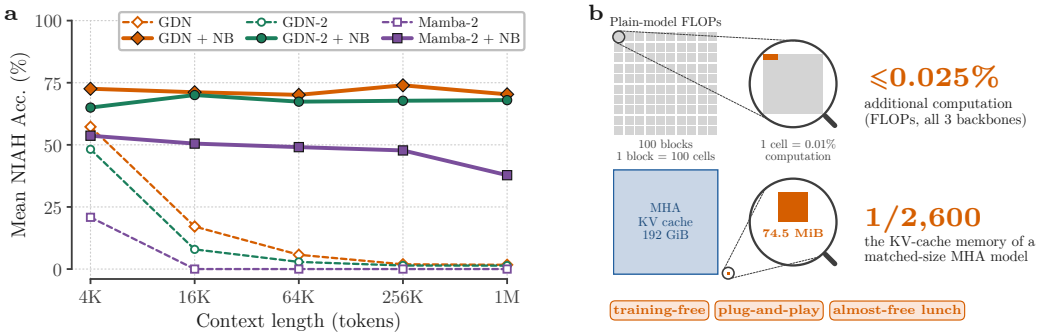


Figure 1: **NoteBank (NB) recovers million-token retrieval with negligible compute overhead**. **a**, Mean accuracy over six RULER retrieval tasks for three pretrained 1.3B recurrent models, without NoteBank (dashed) and with training-free top-2 note replay (solid). **b**, At 1M tokens, NoteBank adds $\leqslant$0.025% layer-level analytical FLOPs across the three backbones (GDN shown). GDN + NoteBank requires 74.5 MiB of persistent context storage, including its recurrent state—approximately 1/2,600 of the 192 GiB MHA KV cache at matched width, depth, and `BF16` precision. Square areas represent storage, with a magnified inset for NoteBank. FLOPs include all evaluated stages at the full-model level and matched output length; they do not measure runtime. Storage excludes model weights and transient workspace.

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.

1 INTRODUCTION

Long-context capability enables language models to reason over extensive documents, conversations, codebases, and interaction histories. Yet making these histories available to a model is costly: full softmax attention (Vaswani et al., 2017; Shazeer, 2019) requires quadratic attention computation during context processing and a key–value cache that grows linearly with sequence length. Linear attention offers a different way to carry information forward. Rather than retaining a separate representation for every past token, it updates a fixed-size recurrent state, connecting attention to recurrent neural networks and fast-weight memory (Schmidhuber, 1991; 1992a; 2025; Katharopoulos et al., 2020; Schlag et al., 2021). This compact representation enables linear-time sequence processing with a constant-size attention state (Sun et al., 2023; Peng et al., 2023; Yang et al., 2024a; Qin et al., 2024). Its practical relevance is reflected in recent large-scale hybrid models such as Griffin (De et al., 2024), Jamba (Lieber et al., 2024), Nemotron-H (NVIDIA, 2025), MiniMax-01 (Li et al., 2025), Qwen3-Next and Qwen3.5 (Team, 2025; Qwen Team, 2026a;b), and Kimi Linear (Kimi, 2025; Moonshot AI, 2026), which combine recurrent linear-attention layers with softmax attention to improve inference efficiency.

This efficiency, however, comes from asking a single recurrent state to serve two roles: ongoing computation and long-term memory (Schmidhuber, 1992b; Wang et al., 2025a). These roles impose competing demands as context grows: computation requires the state to continually transform and compress history, whereas memory requires specific past evidence to remain recoverable even when its relevance emerges only much later. As history grows while the state remains fixed in size, continual compression risks irreversibly discarding evidence before its future relevance is known (Gu & Dao, 2024; GLM-5 Team, 2026). Prior work establishes finite-state limits on exact copying (Jelassi et al., 2024), identifies recency bias and over-smoothing in selective state-space models (Wang et al., 2025b), and documents substantial associative-recall gaps in several efficient alternatives to attention (Arora et al., 2024b). Improved state-update rules (Schlag et al., 2021; Yang et al., 2024b; 2025; Hatamizadeh et al., 2026; Kimi, 2025; Peng et al., 2025; Hu et al., 2025; Siems et al., 2025; Sun et al., 2024; Behrouz et al., 2025; von Oswald et al., 2026) can manage this limited capacity more effectively, but retain the same underlying constraint: the bounded state must still support evolving computation while preserving evidence from an ever-growing history. Together, these limitations reinforce a familiar picture: linear attention is efficient, yet its fixed-state memory appears to impose a fundamental ceiling on long-context capability.

We challenge this apparent ceiling: *poor long-range retention through a recurrent state need not imply an inability to use distant evidence.* We introduce NoteBank, a **training-free, plug-and-play** sparse episodic memory that separates evidence preservation from recurrent computation: selected text spans are archived before any question is observed, addressed using the backbone's native features, and read through its ordinary forward computation via replay. NoteBank augments existing recurrent and hybrid checkpoints without changing backbone weights or architecture, requiring no external embedding model, separately trained retriever, or separate neural reader.

As Figure 1 shows, NoteBank raises Gated DeltaNet's mean accuracy across six RULER (Hsieh et al., 2024) retrieval tasks at one million tokens from 1.7% to 70.3%, with **negligible added computation**: less than 0.026% in analytical layer-level FLOPs. At the same length, GDN + NoteBank requires approximately 1/2600 of the persistent context storage of a full MHA KV cache at matched width, depth, and precision. Beyond controlled retrieval, its benefits extend from 1.3B research models to post-trained Mamba-2-8B (Dao & Gu, 2024) and Qwen3.5-9B (Qwen Team, 2026a), with improvements on LongBench-v1/v2 (Bai et al., 2024; 2025) and MemoryAgentBench (Hu et al., 2026). These results show that the long-context capability of pretrained recurrent models can substantially exceed what their fixed-state memory interface alone exposes.

Our contributions are threefold:

- **Breaking the apparent fixed-state ceiling.** We show that the long-context capability of pretrained recurrent models can substantially exceed what their conventional fixed-state memory interface exposes: poor long-range retention through the recurrent state need not imply an inability to use distant evidence once that evidence is independently preserved and made accessible.
- **Turning pretrained computation into episodic memory.** NoteBank decouples evidence preservation from recurrent computation through question-independent sparse text archiv-

ing, native-feature addressing, and on-demand replay. This ***training-free, plug-and-play*** interface uses the backbone itself to write, address, and read memory, requiring neither retraining nor an external embedding model or separate neural reader.

- **Recovering capability at *near-zero* compute overhead.** Across recurrent and hybrid backbones, NoteBank recovers strong performance from controlled million-token retrieval to natural-document and agent-memory tasks, while adding negligible analytical computation and requiring orders of magnitude less persistent context storage than a matched full-attention KV cache.

2 PRELIMINARY

History compression with linear attention. A broad family of linear-attention models compresses the processed history into recurrent key-value states (Katharopoulos et al., 2020; Schlag et al., 2021). For a single layer and head, the backbone's existing feature transformations produce query and key vectors $\boldsymbol{q}_t, \boldsymbol{k}_t \in \mathbb{R}^{d_k}$ and a value vector $\boldsymbol{v}_t \in \mathbb{R}^{d_v}$ from its input representations at token position t. The key-value pair updates the recurrent memory, while the query reads from it (Yang et al., 2024b; Dao & Gu, 2024):

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

where $\boldsymbol{S}_t \in \mathbb{R}^{d_v \times d_k}$ serves as a compressed memory of the prefix, $\boldsymbol{A}_t \in \mathbb{R}^{d_k \times d_k}$ controls memory retention and transformation, and β_t controls the write strength. Standard additive linear attention (Schmidhuber, 1991; Katharopoulos et al., 2020) sets $\boldsymbol{A}_t = \boldsymbol{I}$ and $\beta_t = 1$; gated (Sun et al., 2023; Yang et al., 2024a; Gu & Dao, 2024; Dao & Gu, 2024; Beck et al., 2024) and delta-rule (Schlag et al., 2021; Yang et al., 2024b; 2025; Hatamizadeh et al., 2026; Kimi, 2025) variants modify these updates. Input scaling is absorbed into this parametrisation; output gates and normalisation are omitted. The same fixed-size state supports both ongoing computation and access to past information: each query reads the history only insofar as it remains encoded in $\boldsymbol{S}_t$.

Problem setup. We consider pretrained recurrent and hybrid language models that process a context before receiving a question. Let $\mathcal{M}_\phi$ be such a model, $\boldsymbol{X} = (x_1, \ldots, x_T)$ the context, and $\boldsymbol{Q} = (q_1, \ldots, q_{|\boldsymbol{Q}|})$ a subsequent question or prompt. The model reads $\boldsymbol{X}$ causally and retains an inference state $\mathcal{S}_T$, consisting of its recurrent states and any auxiliary caches. When $\boldsymbol{Q}$ arrives, it generates a response $\boldsymbol{Y} \sim p_\phi(\cdot \mid \mathcal{S}_T, \boldsymbol{Q})$. Access to the preceding context is therefore limited to what remains in $\mathcal{S}_T$: evidence must be preserved before the model knows which information the question will require.

3 NOTEBANK

3.1 DECOUPLING RECURRENT COMPUTATION FROM ARCHIVAL MEMORY

NoteBank maintains a sparse episodic archive alongside the backbone's unchanged state updates during context processing. Before observing $\boldsymbol{Q}$, it uses native features to select spans, i.e., contiguous text spans (or interchangeably "segments") from the context, and index them for later retrieval. When $\boldsymbol{Q}$ arrives, its native query features retrieve notes, whose text is replayed through the backbone to support generation. After processing t context tokens, the archive is

$$\mathcal{NB}_t = \{(\mathbf{z}_i, \mathcal{A}_i)\}_{i=1}^{N_t}, \qquad \mathcal{A}_i = \{\boldsymbol{\rho}_i^{\ell,h}\}_{(\ell,h)\in\mathcal{L}\times\mathcal{H}}, \tag{2}$$

where N_t is the number of stored notes, $\mathbf{z}_i$ contains a span's original token IDs, and each address $\boldsymbol{\rho}_i^{\ell,h}$ is a compact vector derived from native query features within that span. The sets $\mathcal{L}$ and $\mathcal{H}$ index recurrent layers and heads; address construction and matching are detailed in Sec. 3.2. Archived text persists independently of subsequent state updates.

3.2 A NATIVE MEMORY INTERFACE

Write (top-W). We partition the context into consecutive, non-overlapping chunks of C tokens, carrying the recurrent state across chunk boundaries (Sun et al., 2023; Yang et al., 2024a; Yang & Zhang, 2024). Once a chunk accumulates C tokens, the writer scores its tokens by native-key distinctiveness to select up to W notes:

$$d_t^{\ell,h} = 1 - \left\langle \widehat{\boldsymbol{k}}_t^{\ell,h}, \overline{\boldsymbol{\mu}}_{\mathcal{C}(t)}^{\ell,h} \right\rangle, \qquad d_t = \frac{1}{|\mathcal{L}\times\mathcal{H}|} \sum_{(\ell,h)\in\mathcal{L}\times\mathcal{H}} d_t^{\ell,h}. \tag{3}$$

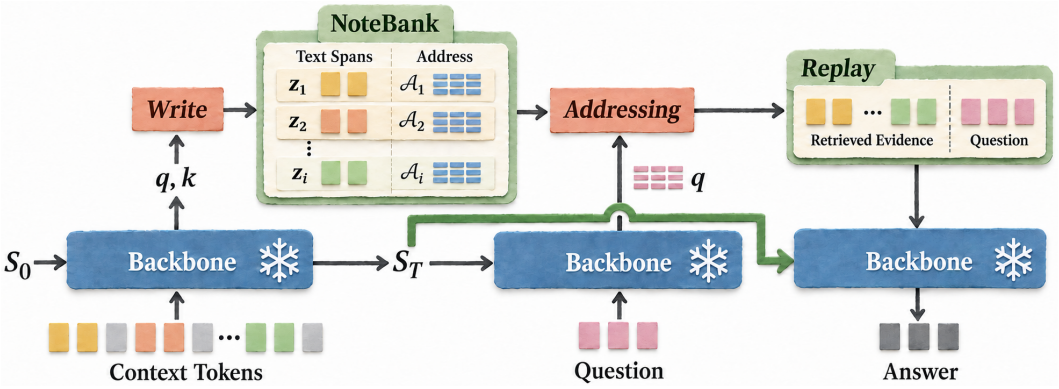


Figure 2: **Overview of NoteBank.** During context encoding, native features "keys" and "queries" (i.e., $\boldsymbol{k}$ and $\boldsymbol{q}$) guide span selection and indexing; each span $\mathbf{z}_i$ is stored with addresses $\mathcal{A}_i$. Taking the retained context state $\boldsymbol{S}_T$, the backbone encodes the question to obtain queries for retrieving relevant spans. The native forward pass starts from the same $\boldsymbol{S}_T$ and processes the retrieved evidence and then the question, and generates the answer. Backbone parameters remain frozen throughout.

Here $\mathcal{C}(t)$ contains the positions in the chunk containing t, and $\overline{\boldsymbol{\mu}}^{\ell,h}_{\mathcal{C}(t)}$ is the normalised mean of the keys in that chunk. Hats $\widehat{\cdot}$ denote ℓ_2 normalisation throughout this paper.

To form each note, we score candidate windows, i.e., short contiguous sequences of tokens, by summing d_t over their positions. We choose the highest-scoring window and use the sentence containing its central token as the starting segment, then expand it with adjacent sentences subject to the B-token limit. Any sentence overlapping the selected segment is excluded from subsequent selections. For each selected text segment with token positions I_i, we store its original token IDs $\mathbf{z}_i$ and construct one address per layer-head channel. Each channel selects the token with the highest key-distinctiveness score and stores a compact version of its native query vector:

$$t_i^{*,\ell,h} = \arg\max_{t \in I_i} d_t^{\ell,h}, \qquad \boldsymbol{\rho}_i^{\ell,h} = \mathrm{compact}\left(\boldsymbol{q}^{\ell,h}_{t_i^{*,\ell,h}}\right) \in \mathbb{R}^{d_\rho}. \tag{4}$$

Thus, each channel indexes the segment using the query at its most distinctive key position; different channels may select different positions. The fixed mapping compact is a non-trainable feature map for constructing compact addresses; setting compact to identity presents full native-query addresses. Full details of address construction is provided in App. B.3.

Addressing (top-R). When $\boldsymbol{Q}$ arrives, we process it from a copy of the retained inference state $\mathcal{S}_T$ to obtain native query features. For each question position $\tau \in \{1, \ldots, |\boldsymbol{Q}|\}$, we construct $\boldsymbol{u}_\tau^{\ell,h} = \mathrm{compact}(\boldsymbol{q}_\tau^{\ell,h})$ using the same mapping as the stored addresses. Each layer-head channel scores every stored note $i \in \{1, \ldots, N_T\}$ by the maximum cosine similarity between its address and the question's query features (Khattab & Zaharia, 2020), then votes for its highest-scoring note:

$$s_i^{\ell,h} = \max_{1 \le \tau \le |\boldsymbol{Q}|} \left\langle \widehat{\boldsymbol{u}}_\tau^{\ell,h}, \widehat{\boldsymbol{\rho}}_i^{\ell,h} \right\rangle, \qquad V_i = \sum_{(\ell,h) \in \mathcal{L} \times \mathcal{H}} \mathbf{1}\left[i = \arg\max_j s_j^{\ell,h} \right]. \tag{5}$$

We retrieve up to R notes with the most votes, with a preference of earlier-inserted notes when vote counts tie. We then concatenate their token IDs in original document order, determined by each note's starting token position, to form the replay sequence $\boldsymbol{R}$. The temporary state used to process $\boldsymbol{Q}$ is discarded after retrieval.

Readout with native replay. After addressing, the backbone reads the retrieved text through its native forward computation. Let $\mathcal{S}_{\mathrm{read}}$ be a fresh copy of the retained state $\mathcal{S}_T$, and $F_\phi(\mathcal{S}, \boldsymbol{X}_*)$ denote the backbone's state update when processing a token sequence $\boldsymbol{X}_*$. We replay $\boldsymbol{R}$, then process $\boldsymbol{Q}$ again to generate the response:

$$\mathcal{S}_{\boldsymbol{R}} = F_\phi(\mathcal{S}_{\mathrm{read}}, \boldsymbol{R}), \qquad \boldsymbol{Y} \sim p_\phi(\cdot \mid \mathcal{S}_{\boldsymbol{R}}, \boldsymbol{Q}). \tag{6}$$

Replay introduces at most $R \times B$ tokens and traverses the full backbone, including attention blocks under the configured policy in hybrid models. This readout requires neither additional training nor a

separate neural reader. Because it follows native forward computation, backbone-supported parallel algorithms (Yang & Zhang, 2024) apply directly, without NoteBank-specific customisation.

To improve the model's use of replayed evidence on downstream tasks, we optionally train lightweight LoRA adapters (Hu et al., 2022) while keeping pretrained weights frozen. Adapters are disabled during context processing and retrieval, leaving writing and addressing unchanged. They are enabled throughout replay and subsequent question answering. See App. B.4 for more details.

3.3 Memory and Computational Overhead

We analyse NoteBank's additional costs relative to the same backbone and attention configuration.

Memory. Let $J = |\mathcal{L} \times \mathcal{H}|$ be the number of indexed recurrent layer-head channels. Writing at most W notes per completed C-token chunk gives:

$$N_T \le W\lfloor T/C \rfloor, \qquad \text{Storage}_{\mathcal{NB}} = O(N_T(B + Jd_\rho)) \,. \tag{7}$$

Each note stores at most B token IDs and $J = |\mathcal{L} \times \mathcal{H}|$ compact addresses of dimension d_ρ. The storage of $\mathcal{NB}$ archive *grows linearly with history length* at a rate controlled by these budgets, without enlarging the recurrent state.

Computation. Writing reuses native features, adding $O(T)$ work for fixed chunk size and model dimensions. Addressing requires one additional forward pass over the question, followed by exhaustive matching at cost $O(|\boldsymbol{Q}|N_T J d_\rho)$, layer-head channel voting, and top-R selection. Readout adds at most $R \times B$ replay tokens before question processing and generation. Thus, archive storage and exhaustive matching *grow linearly with history length*, while *the replay budget remains constant*; replay computation depends on the backbone and attention policy.

4 Recovering Long-Context Capability

Can pretrained recurrent models recover strong long-context capability when past evidence remains accessible beyond their recurrent states? We investigate this question through long-range retrieval at increasing context lengths, real-world tasks, and controlled restrictions on historical access.

Setup. We evaluate *publicly available checkpoints* of GDN and GDN-2 (1.3B), Mamba-2 (1.3B and 8B), and Qwen3.5 (9B). For Mamba-2, recurrent-state rescaling is necessary for the plain baseline to avoid numerical failure during long-context inference. NB-TF is training-free while NB-LoRA adapts the read stage using lightweight adapters trained on *fewer than 9.3M tokens of general language data*, with one adapter set per backbone shared across downstream tasks. Both variants construct notes *without observing the evaluation question*. Our evaluation covers six RULER-NIAH tasks (Hsieh et al., 2024), LongBench-v1 and v2 (Bai et al., 2024; 2025), and MemoryAgentBench (Hu et al., 2026). We refer to App. C for more experimental details.

4.1 Training-free long-range retrieval across architectures and lengths

We first ask: how much long-range retrieval can be recovered by decoupling recurrent computation from archival memory, without updating the model's weights? We compare plain, full-prompt repetition (JRT) (Arora et al., 2024c), and NB-TF on GDN, GDN-2, and Mamba-2 using the same checkpoint within each architecture. Evaluation covers three single-needle tasks, `S-NIAH-1/2/3`, and three `multi-key`, `multi-query`, and `multi-value` tasks, 100 samples per task. Document lengths range from 4K to 1M tokens; JRT processes approximately twice this input length through repetition. NoteBank writes its archive before observing the question.

Results &. Discussion. Figure 3 reports mean accuracy across the six tasks. Plain and JRT approach zero accuracy as context length grows, whereas NB-TF maintains roughly 70% accuracy on GDN and GDN-2. Mamba-2 declines more noticeably but retains substantial accuracy at 1M tokens. At 1M, NB-TF exceeds Plain by 68.6, 66.6, and 37.8 percentage points on GDN, GDN-2, and Mamba-2, respectively. The same frozen backbones thus recover substantial retrieval capability through NoteBank's memory, while full-prompt repetition yields little improvement at long contexts. We next test whether these gains translate to real-world tasks.

4.2 From synthetic retrieval to real-world tasks

To test whether NoteBank improves performance beyond synthetic retrieval, and whether read adaptation helps models use the archived evidence, we evaluate GDN and Mamba-2 on LongBench

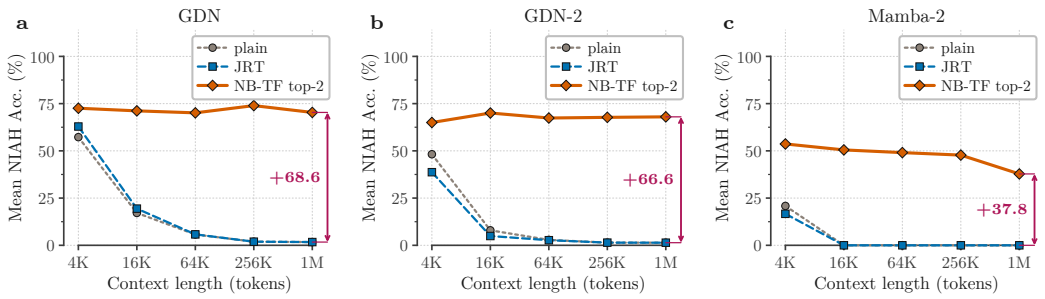


Figure 3: **Recovering long-range retrieval *without training*.** NoteBank (NB-TF top-2) substantially improves retrieval over plain forward pass and full-prompt repetition (JRT), using the same frozen backbone. Scores are averaged over six RULER retrieval tasks, including single- and multi-passage retrieval. See App. D.1 for per-task results.

Backbone	Method	Single-doc QA	Multi-doc QA	Summarisation	Few-shot	Overall
Transformer†	Plain	13.40	18.97	8.43	23.40	16.05
GDN	Plain	6.88	7.02	13.99	34.08	15.49
	JRT	7.72	6.63	**14.79**	33.56	15.67
	NB-TF	6.66	7.33	13.52	36.76	16.07
	NB-LoRA	**19.06**	**20.48**	11.37	**41.22**	**23.03**
Mamba-2	Plain	2.22	2.10	11.27	11.56	6.79
	JRT	2.39	1.89	9.21	8.34	5.12
	NB-TF	5.98	5.36	**16.51**	25.88	13.43
	NB-LoRA	**17.84**	**15.54**	7.46	**27.52**	**17.10**

Table 1: **From retrieval to real-world long-context tasks.** All backbones have 1.3B parameters. Scores are macro-averaged over evaluated LongBench-v1 tasks, within each category and overall. NB-TF requires no additional training; NB-LoRA adapts the read stage. †Reference results aggregated from Behrouz et al. (2026), whose LongBench models were trained with 16K contexts. Bold marks the best score within each recurrent backbone.

Benchmark	Backbone	Plain	+ NB-TF	Δ
MemoryAgentBench	Mamba2-8B	0.0	14.7	**+14.7**
	Qwen3.5-9B	55.7	56.8	**+1.1**
LongBench-v2	Qwen3.5-9B	46.1	49.3	**+3.2**

Table 2: ***Plug-and-play* gains on larger models without training.** We report task-macro-averaged scores on the 2,100-question QA subset of MemoryAgentBench and accuracy on the 375-question QA/dialogue subset of LongBench-v2. NB-TF uses top-4 and top-8 retrieval, respectively. All Qwen evaluations use YaRN (official long-context setup). Δ denotes the gain (%).

12 tasks spanning `single-doc QA`, `multi-doc QA`, `summarisation`, and `few-shot learning`. Within each backbone, we compare Plain, JRT, NB-TF, and NB-LoRA on the same examples. Both NoteBank variants use top-1 replay for GDN and top-4 for Mamba-2; NB-LoRA adapts only the read stage, leaving note writing and addressing unchanged. We also include published Transformer results from Behrouz et al. (2026) as a reference. Scores are task averages within each category and overall.

Results and discussion. Table 1 shows that NB-TF improves over Plain in six of eight backbone-category comparisons, raising overall scores from 15.49 to 16.07 on GDN and from 6.79 to 13.43 on Mamba-2. GDN with NB-TF reaches an overall score comparable to the Transformer reference (16.05), without additional training. Read adaptation strengthens the gains: NB-LoRA raises overall scores to 23.03 and 17.10, respectively, both above the Transformer reference, with substantial improvements in single- and multi-document QA over the corresponding Plain backbones. We hypothesise that replay can introduce distracting evidence, and that read adaptation helps the model

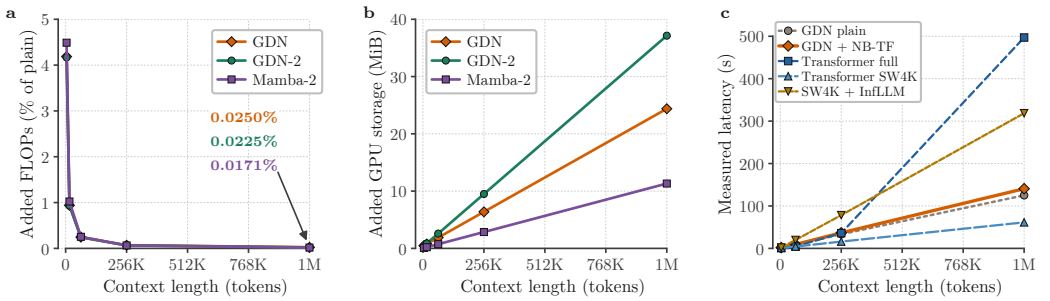


Figure 4: **Efficiency of training-free NoteBank.** (a) Relative analytical FLOPs overhead, decreasing with context length as fixed costs are amortised. (b) Marginal persistent GPU storage grows linearly with context. (c) Measured inference latency on an H100, with batch size 1 and 32 generated tokens. NB uses top-2 replay; InfLLM use the official default top-96.

use replayed evidence more selectively; we investigate this explanation in Section 6. The benefit of read adaptation remains task-dependent, as NB-LoRA underperforms NB-TF on summarisation.

4.3 Plug-and-play on off-the-shelf LLMs

Can NoteBank improve larger off-the-shelf LLMs, including a strong post-trained hybrid, without additional training? We evaluate Mamba-2-8B and Qwen3.5-9B on selected MemoryAgentBench tasks, and Qwen3.5-9B on the 375-question QA/dialogue subset of LongBench-v2. Within each backbone, we compare Plain and NB-TF using the same checkpoint and examples. NB-TF uses top-4 replay on MemoryAgentBench and top-8 on LongBench-v2; both methods use YaRN for Qwen. Retrieval-budget comparisons and task-level results are provided in the App. D.1.

Results and discussion. Table 2 shows gains across all three model-benchmark pairs. On MemoryAgentBench, plain Mamba-2 fails completely while, with NB-TF, it improves from 0.0 to 14.7. Qwen improves from 55.7 to 56.8 on MemoryAgentBench and from 46.1 to 49.3 on LongBench v2, demonstrating benefits even for a strong post-trained hybrid with existing long-context capabilities. We attribute Mamba-2's complete failure in this setting to long-context state instability (detailed in Appendix D.1). Together, these results support NoteBank as a plug-and-play memory extension across recurrent and hybrid architectures, without additional training.

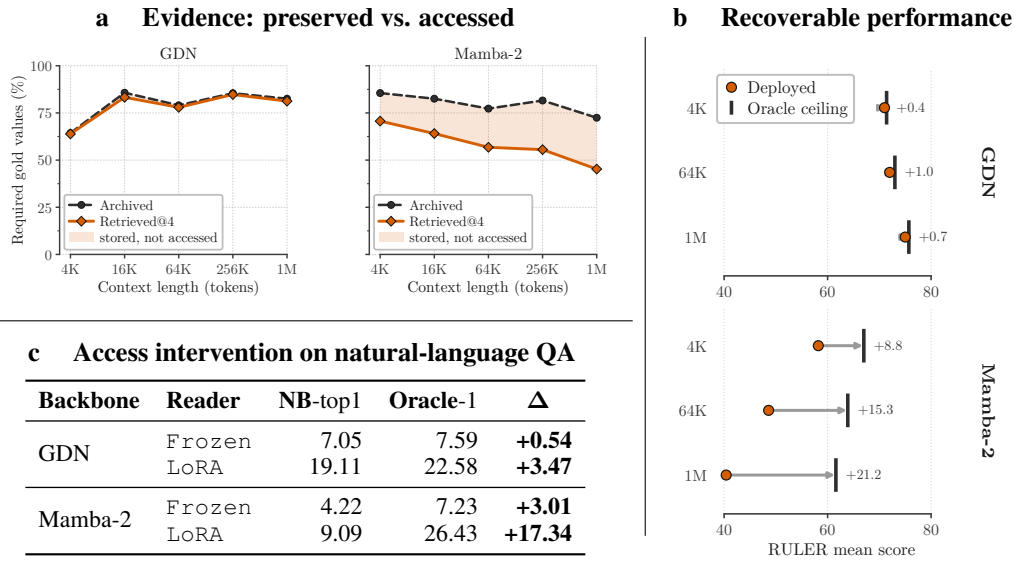
5 Almost a Free Lunch?

Having shown that NoteBank recovers substantial long-context capability, we now ask: at what cost? We quantify its incremental computation and storage analytically, then measure end-to-end latency and peak GPU memory, relating these costs to retrieval quality (Figure 4; Table 7).

Negligible marginal computation. Relative FLOPs overhead decreases with context length as fixed encoding and replay costs are amortised over longer backbone computation (Figure 4a). Although writing and routing grow linearly with context, reusing native features keeps their cost small: at 1M tokens, NoteBank top-2 adds only 0.0171%-0.0250% to full-model analytical FLOPs across three backbones, with routing costing just 0.50–1.72 GFLOPs. Accounting covers end-to-end inference with matched output lengths (App. C.5, Figure 7).

Compact persistent storage. NoteBank stores selected token IDs on the CPU and compact native addresses on the GPU, retaining the backbone's fixed-size recurrent and convolution states. The added persistent GPU storage grows linearly with context length (Figure 4b). At 1M tokens, GDN adds 24.4 MiB of GPU addresses and helper tensors, plus approximately 1.0 MiB of CPU text and metadata accounted for in compact form. Including backbone states, total persistent context storage is 74.5 MiB, approximately $1/2600$ of a theoretical `BF16` full MHA KV cache matched in context length, width, and depth. Accounting excludes model weights, transient allocations, and Python-object overhead (App. C.7).

Practical runtime and memory We measure end-to-end latency and peak GPU memory on an H100 with batch size 1 and 32 generated tokens. Figure 4c shows that NoteBank increases GDN's latency by 10.6%-13.2% across evaluated lengths. At 1M tokens, peak GPU memory rises from 2.93



c Access intervention on natural-language QA

Backbone	Reader	NB-top1	Oracle-1	Δ
GDN	`Frozen`	7.05	7.59	**+0.54**
	`LoRA`	19.11	22.58	**+3.47**
Mamba-2	`Frozen`	4.22	7.23	**+3.01**
	`LoRA`	9.09	26.43	**+17.34**

Figure 5: **Evidence preservation, access, and utilisation in NoteBank.** **a**, Gold-value coverage in the archive (*Archived*) and top-4 retrieved notes (*Retrieved@4*); shading marks stored but unretrieved values. **b**, Native vs. oracle routing on RULER, with archive, reader, and replay budget fixed. **c**, The same intervention on four LongBench QA tasks (top-1; frozen/LoRA readers). Oracle notes contain reference answers, a proxy for relevance rather than guaranteed supporting evidence.

to 3.27 GiB (Table 7). Thus, negligible marginal arithmetic translates into modest, nonzero runtime overhead. Quality-cost comparisons with Transformer and InfLLM, along with measurement protocols are detailed in App. C.5. Together, these results show that NoteBank extends pretrained linear attention models with substantially stronger long-context retrieval, requiring negligible added arithmetic, compact storage, and modest runtime overhead—*an almost free lunch*.

6 Understanding NoteBank: Evidence Access and Utilisation

We examine what limits NoteBank's performance by tracing evidence preservation and retrieval, intervening on note selection, and comparing frozen and adapted readers to distinguish the contributions of memory access and read adaptation.

6.1 What is preserved, and what is retrieved?

Preserving evidence does not guarantee its retrieval. We therefore measure the fractions of required gold values preserved in the archive and recovered by top-4 retrieval, respectively. Figure 5a shows that retrieval closely tracks archival coverage for GDN, whereas Mamba-2 leaves substantially more stored values unretrieved. Holding the archive, reader, and replay budget fixed, oracle note selection raises GDN's score from 75.0 to 75.7 at 1M tokens, but Mamba-2's from 40.4 to 61.6 (Figure 5b). This intervention reveals substantial routing headroom for Mamba-2, with smaller gains for GDN.

We extend this diagnostic to four LongBench QA tasks (`hotpotqa`, `2wikimqa`, `musique`, `qasper`), using archived spans containing a reference answer as a proxy for relevant evidence. With the archive and reader fixed, we compare native top-1 retrieval with oracle selection of a single answer-bearing note ("Oracle-1" in Figure 5c). Replaying just this one note instead improves Mamba-2's macro score by 3.01 points with a frozen reader and by 17.34 points with LoRA; GDN gains 0.54 and 3.47 points, respectively. Note that answer containment does not guarantee sufficient supporting evidence, so these results are a proxy diagnostic rather than a performance upper bound. Together, these interventions show that preserved information can remain unaccessed and that the benefit of improved access depends on the reader.

6.2 How much does routing matter in practice?

We next test whether alternative routers can realise this headroom, and at what cost. We hold the archive, reader, and top-1 replay budget fixed and replace only the router. As shown in Table 3, on RULER at 1M tokens, native routing outperforms random selection, mean-pooled input embeddings,

Routing Signal	Auxiliary Retriever	RULER @ 1M		LongBench	
		GDN	Mamba-2	GDN	Mamba-2
Random	`None`	1.7	0.0	21.96	13.83
Input embedding	`None`	30.8	20.8	22.40	**16.70**
Native (NB)	`None`	**59.8**	**37.6**	**23.03**	14.25
BM25	`Lexical index`	**64.0**	**52.7**	23.50	**18.66**
BGE	`External encoder`	45.4	34.0	**23.77**	17.24

Table 3: **Effect of routing strategy.** Scores average 6 RULER tasks at 1M tokens and 12 evaluated LongBench tasks, with LoRA readers fixed across routers for each backbone and benchmark. Native (NB) uses feature matching and voting; input embeddings are mean-pooled. Auxiliary retrievers add a lexical index or encoder; `None` does not imply zero routing cost.

Memory Access	RULER @ 1M		LongBench	
	Frozen	LoRA	Frozen	LoRA
`w/o` NoteBank	1.7	1.7	15.49	22.72
`w/` NoteBank	**59.8**	**58.4**	**16.07**	**23.03**

Table 4: **Separating memory access from read adaptation.** GDN scores with and without NoteBank, using frozen or LoRA-adapted readers. Scores average six RULER retrieval tasks at 1M tokens and twelve LongBench tasks. NoteBank uses top-1 replay; adapters are trained independently with and without replay using matched data and training budgets.

and BGE on both backbones. BM25 achieves higher scores, raising Mamba-2 from 37.6 to 52.7, consistent with the routing headroom identified in Section 6.1. On LongBench with a fixed LoRA reader, differences are modest for GDN but larger for Mamba-2, where BM25 improves the score from 14.25 to 18.66. For GDN at 1M tokens, the default NB router's query encoding and routing take 26 ms, versus 87 ms for CPU BM25 in our implementation (App. D.1, Table 8). These comparisons show that the default native routing provides effective retrieval with low measured routing latency and no auxiliary encoder, while gains from alternative routers motivate further exploration of the retrieval quality-cost trade-off within NoteBank.

6.3 What does read adaptation contribute?

To distinguish evidence recovery from adaptation gains, we separate memory-access gains from adaptation gains using GDN with and without NoteBank, each with frozen or LoRA readers (Table 4). Adapters are trained separately with and without replay using the same data and training budget; NoteBank uses top-1 retrieval. On RULER at 1M tokens, adaptation without NB leaves the score at 1.7, whereas NB-TF reaches 59.8. Adding LoRA yields 58.4, showing that this long-range retrieval recovery does not require read adaptation. On LongBench, adaptation improves scores from 15.49 to 22.72 without NB and from 16.07 to 23.03 with NB. The adapted models differ by only 0.31 points, indicating that adaptation accounts for most GDN's LongBench improvement in this setting. These controls distinguish two contributions: NoteBank enables long-range retrieval with frozen weights, while task adaptation drives most of the observed LongBench gains on GDN.

7 Discussion and Limitations

Our results challenge the view that linear attention must sacrifice long-context capability for efficiency, showing that separating recurrent computation from memorisation enables substantially stronger long-context performance at little additional cost. NoteBank realises this separation with sparse notes, native-feature addressing, and backbone replay. This ***training-free, plug-and-play*** extension offers an ***almost free lunch***: 37.8-68.6 percentage-point gains at 1M-token retrieval across three linear attention architectures, with negligible marginal arithmetic, storage, and modest runtime overhead. Benefits extend to real-world tasks and a strong post-trained hybrid. This practical architecture leaves ample scope to improve writing, routing, and reading. Our interventions reveal opportunities to improve routing and evidence utilisation, with gains varying across tasks and backbones. Training-free effectiveness establishes compatibility with existing models, while selectively training writing, routing, or reading may improve evidence preservation, access, and utilisation.

AI USE STATEMENT

Generative AI tools were used to assist with code implementation and debugging, as well as language editing and presentation of the manuscript. The core research idea was developed by the authors. AI-assisted code was reviewed and tested by the authors before use in the experiments, and all AI-assisted manuscript content was manually reviewed and revised for technical correctness and consistency with the reported results. The authors take full responsibility for the final content of the paper, including all code, experiments, analyses, and claims produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed information to support reproducibility in the appendix, including the complete NoteBank configuration, implementation details, benchmark and evaluation protocols, read-adaptation settings, and efficiency measurement procedures. We additionally report per-task and extended experimental results to facilitate verification of the main findings.

REFERENCES

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/448fc91f669c15d10364ee01d512cc10-Abstract-Conference.html.

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, James Zou, Atri Rudra, and Christopher Re. Simple linear attention language models balance the recall-throughput tradeoff. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 1763–1840. PMLR, 21–27 Jul 2024b. URL https://proceedings.mlr.press/v235/arora24a.html.

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*, 2024c. URL https://arxiv.org/abs/2407.05483.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 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.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664, 2025.

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. *arXiv preprint arXiv:2405.04517*, 2024.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *Advances in Neural Information Processing Systems*, 2025. URL https://papers.nips.cc/paper_files/paper/2025/file/a4ca07aa108036f80cbb5b82285fd4b1-Paper-Conference.pdf.

Ali Behrouz, Zeman Li, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. Memory caching: RNNs with growing memory. *arXiv preprint arXiv:2602.24281*, 2026. URL https://arxiv.org/abs/2602.24281.

Kyunghyun Cho. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

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

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

Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models. *arXiv preprint arXiv:2402.19427*, 2024.

GLM-5 Team. GLM-5: From Vibe Coding to Agentic Engineering. *arXiv preprint arXiv:2602.15763*, 2026.

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

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

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8): 1735–1780, 1997.

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? In *First Conference on Language Modeling (COLM)*, 2024.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Jiaxi Hu, Yongqi Pan, Jusen Du, Disen Lan, Xiaqiang Tang, Qingsong Wen, Yuxuan Liang, and Weigao Sun. Comba: Improving Bilinear RNNs with Closed-Loop Control. In *Neural Information Processing Systems*, 2025.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions. In *International Conference on Learning Representations (ICLR)*, 2026.

Samy Jelassi, David Brandfonbrener, Sham M. Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 21502–21521. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/jelassi24a.html.

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.

Omar Khattab and Matei Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 39–48, 2020.

Kimi. Kimi linear: An expressive, efficient attention architecture, 2025. URL https://arxiv.org/abs/2510.26692.

Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. A human-inspired reading agent with gist memory of very long contexts. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=vRHrqXVTiQ.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230-Abstract.html.

Aonian Li, Bangwei Gong, Bo Yang, Boji Shan, Chang Liu, Cheng Zhu, Chunhao Zhang, Congchao Guo, Da Chen, Dong Li, et al. MiniMax-01: Scaling Foundation Models with Lightning Attention. *arXiv preprint arXiv:2501.08313*, 2025.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid Transformer-Mamba language model. *arXiv preprint arXiv:2403.19887*, 2024. URL https://arxiv.org/abs/2403.19887.

Peng Lu, Jerry Huang, Qiuhao Zeng, Xinyu Wang, Boxing Chen, Philippe Langlais, and Yufei Cui. Mamba modulation: On the length generalization of mamba, 2025. URL https://arxiv.org/abs/2509.19633.

Moonshot AI. Kimi K2.6. https://huggingface.co/moonshotai/Kimi-K2.6, 2026.

Niklas Muennighoff, Hongjin Su, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. Generative representational instruction tuning. *arXiv preprint arXiv:2402.09906*, 2024. URL https://arxiv.org/abs/2402.09906.

NVIDIA. Nemotron-H: A Family of Accurate and Efficient Hybrid Mamba-Transformer Models. *arXiv preprint arXiv:2504.03624*, 2025.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, et al. RWKV: Reinventing RNNs for the Transformer Era. In *Findings of the Association for Computational Linguistics: EMNLP*, 2023.

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Haowen Hou, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, Nathan Wilce, Johan S Wind, Tianyi Wu, Daniel Wuttke, and Christian Zhou-Zheng. RWKV-7 "Goose" with Expressive Dynamic State Evolution. In *Conference on Language Modeling*, 2025.

Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. HGRN2: Gated Linear RNNs with State Expansion. In *Conference on Language Modeling*, 2024.

Qwen Team. Qwen3.5: Towards Native Multimodal Agents, 2026a. URL https://qwen.ai/blog?id=qwen3.5.

Qwen Team. Qwen3.6-Plus: Towards Real World Agents, 2026b. URL https://qwen.ai/blog?id=qwen3.6.

Ricardo Buitrago Ruiz and Albert Gu. Understanding and improving length generalization in recurrent models, 2025. URL https://arxiv.org/abs/2507.02782.

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 Recurrent Nets. Technical Report FKI-147-91, Institut für Informatik, Technische Universität München, 1991.

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

Jürgen Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural computation*, 4(2):234–242, 1992b.

Jürgen Schmidhuber. The 1991 Unnormalized Linear Transformer (ULTRA), 2025. URL https://people.idsia.ch/~juergen/1991-unnormalized-linear-transformer.html. AI Blog.

Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019. URL https://arxiv.org/abs/1911.02150.

Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazzi. DeltaProduct: Improving State-Tracking in Linear RNNs via Householder Products. In *Neural Information Processing Systems*, 2025.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): RNNs with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024. URL https://arxiv.org/abs/2407.04620.

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.

Qwen Team. Qwen3-Next: Towards Ultimate Training & Inference Efficiency, 2025. URL https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd&from=research.latest-advancements-list.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

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*, 2026.

Ke Alexander Wang, Jiaxin Shi, and Emily B Fox. Test-Time Regression: A Unifying Framework for Designing Sequence Models with Associative Memory. *arXiv preprint arXiv:2501.12352*, 2025a.

Peihao Wang, Ruisi Cai, Yuehao Wang, Jiajun Zhu, Pragya Srivastava, Zhangyang Wang, and Pan Li. Understanding and mitigating bottlenecks of state space models through the lens of recency and over-smoothing. In *International Conference on Learning Representations*, volume 2025, pp. 82314–82342, 2025b.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. InfLLM: Training-Free Long-Context Extrapolation for LLMs with an Efficient Context Memory. In *Neural Information Processing Systems*, 2024.

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. In *International Conference on Machine Learning*, 2024a.

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

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

Ming Zhang, Kaisen Yang, Shu Yu, Ermo Hua, Ning Ding, Xia Hu, Bowen Zhou, Chaochao Lu, and Youbang Sun. MARCH: Scaling recurrent memory with content-routed state anchors. *arXiv preprint arXiv:2608.12435*, 2026. URL https://arxiv.org/abs/2608.12435.

Yanjun Zhao, Ruizhong Qiu, Tianxin Wei, Yuanchen Bei, Zhining Liu, Lingjie Chen, Ismini Lourentzou, Hanghang Tong, and Jingrui He. ReContext: Recursive evidence replay as LLM harness for long-context reasoning. *arXiv preprint arXiv:2607.02509*, 2026. URL https://arxiv.org/abs/2607.02509.

APPENDIX CONTENTS

A Related Works **15**

B Algorithmic Details **16**

B.1 Algorithmic overview 16

B.2 Write operation details 16

B.3 Native address construction 17

B.4 Replay readout and optional adaptation 18

C Experimental Details **18**

C.1 Backbones 18

C.2 Benchmarks and evaluation protocols 19

C.3 NoteBank and baseline configurations 19

C.4 Adaptation setup 20

C.5 Implementation and efficiency measurement 20

C.6 Added FLOPs Accounting 21

C.7 Persistent Memory Accounting 22

D Additional Results **22**

D.1 Complementary results 22

D.2 Ablation study 26

A RELATED WORKS

Linear and Recurrent Long-Context Models. Recurrent sequence models (Schmidhuber, 1992a;b; Hochreiter & Schmidhuber, 1997; Cho, 2014; Beck et al., 2024) reduce the cost of quadratic self-attention by compressing the input prefix into a state whose size is independent of sequence length. Mambas (Gu & Dao, 2024; Dao & Gu, 2024) use selective state-space updates, while DeltaNet (Schlag et al., 2021; Yang et al., 2024b) and Gated DeltaNet (Yang et al., 2025) maintain associative memories through delta-rule updates. With fixed state size and numerical precision, a single pass cannot preserve arbitrarily growing histories losslessly. Reliable recall therefore depends on what the state retains and how stored associations interfere (Schlag et al., 2021; Arora et al., 2024a;c). Hybrid architectures combine recurrent layers with attention to improve access to token-level representations (Lieber et al., 2024; Yang et al., 2024b). In particular, hybrid variants of Gated DeltaNet and Gated DeltaNet-2 (Yang et al., 2025; Hatamizadeh et al., 2026) incorporate sliding-window attention for local interactions and report improvements on evaluated retrieval tasks. In these sliding-window hybrids, direct attention access remains confined to the local window, while information from more distant inputs is carried through the recurrent computation.

Memory Extension for Recurrent Models. Recurrent memory methods develop learnable state updates and retain additional historical memory for later reading. TTT layers (Sun et al., 2024) treat the recurrent hidden state as the weights of a learner updated through a self-supervised objective, while Titans (Behrouz et al., 2025) learns a neural long-term memory through online updates with momentum and forgetting. These methods change how history is encoded into a state of fixed size. Memory Caching (Behrouz et al., 2026) additionally preserves multiple memory states, allowing effective memory capacity to grow with sequence length; it considers both continuous-recurrence checkpoints and independently initialized segment memories, and evaluates caching on

Titans. MARCH (Zhang et al., 2026) caches cumulative recurrent-state anchors and uses content-dependent routing to combine historical readouts with the current-state output. HOLA (Cui, 2026) augments the recurrent state with a bounded cache of key-value pairs selected according to the recurrent update magnitude. These caching methods differ in how much history they retain and whether they store compressed states or selected key-value pairs. Their retrieval operations read the retained states or values using current queries, without re-encoding the corresponding raw inputs.

Retrieval and Replay. Retrieval-based methods access external corpora or prior inputs and incorporate the retrieved information through representation fusion or renewed text processing. Retrieval-augmented generation (RAG) can provide textual evidence that need not have appeared in the current input history (Lewis et al., 2020). Standard dense-retrieval pipelines select passages by matching query and passage embeddings and incorporate the retrieved text into the context for generation. Retrieval can also use the generation model's own representations: GritLM (Muennighoff et al., 2024) unifies embedding and generation within a single language model, using a contrastive embedding objective to support corpus retrieval. Its corpus embeddings are produced through dedicated embedding passes, rather than extracted from the causal processing of the current input history. Another line of work revisits previously processed inputs. ReadAgent (Lee et al., 2024) selects original pages through reasoning over textual gist summaries, while ReContext (Zhao et al., 2026) uses model-internal attention signals to select and replay evidence spans from the available prompt. Just Read Twice (Arora et al., 2024c) analyses how input order affects recall memory requirements and studies prompt repetition and non-causal prefix processing to exploit information unavailable during the initial causal pass.

Our approach stores raw historical tokens together with retrieval keys derived from model states during their initial causal processing. During subsequent causal computation, the current retrieval query selects tokens to replay through the model's input-processing path, allowing a new encoding after that query becomes available.

B Algorithmic Details

B.1 Algorithmic overview

Algorithm 1 summarises the construction and use of NoteBank. During context processing, the writer maintains the backbone state and builds an archive independently of the subsequent question. At query time, retrieval and readout branch from separate copies of the retained state: the question is first processed to select notes, after which the retrieved text and the question are processed on the answer branch. For clarity, the algorithm abstracts segment (or interchangeably, "span") selection, address construction, and matching into named operations. Chunk-boundary handling and sentence-level selection rules are specified in Appendix B.2.

B.2 Write operation details

The writer operates at completed chunk boundaries, without observing the subsequent question. Within each chunk $\mathcal{C}_i$, we first ℓ_2-normalise each native key, average the resulting unit vectors, and normalise the mean:

$$\overline{\boldsymbol{\mu}}_i^{\ell,h} = \widehat{\frac{1}{|\mathcal{C}_i|} \sum_{t \in \mathcal{C}_i} \widehat{\boldsymbol{k}}_t^{\ell,h}}. \tag{8}$$

We average distinctiveness scores across layer–head channels to select text segments, and retain the per-channel scores to construct their addresses. To capture evidence that spans a chunk boundary, we extend the candidate region with the last 32 tokens of the preceding chunk. These tokens keep the scores and query features computed in their original chunk; their scores are not recomputed for the extended region. By default, we use this 32-token overlap and a 48-token scoring window.

Within each candidate region, the writer scores sliding windows by summing their token distinctiveness and selects the highest-scoring window. If the region is shorter than the configured window length, the window covers the entire region. We split the observed text at sentence-final punctuation and line breaks, and use the sentence containing the selected window's central token as the initial span. We then expand the span by checking the immediately adjacent sentences, first on the left and

Algorithm 1 NoteBank: Write, Address, and Replay

Input: Frozen backbone $\mathcal{M}$; context $\boldsymbol{X}$; question $\boldsymbol{Q}$.
HyperParams: Chunk size C; write budget W; note length limit B; retrieval budget R.
Output: Generated response $\boldsymbol{Y}$.

1. Write *Before observing the question*

$\mathcal{S} \leftarrow \mathcal{M}_\phi.\texttt{init_state}$; $\mathcal{NB} \leftarrow [\,]$
for each chunk $\boldsymbol{X}_\mathcal{C}$ in $\texttt{chunks}(\boldsymbol{X}, C)$ **do**
  $\mathcal{S}, \{\boldsymbol{k}_\mathcal{C}\}, \{\boldsymbol{q}_\mathcal{C}\} \leftarrow \mathcal{M}_\phi.\texttt{forward}(\mathcal{S}, \boldsymbol{X}_\mathcal{C})$
  $\{d_\mathcal{C}\} \leftarrow \textsc{DistinctivenessMeasure}(\{\boldsymbol{k}_\mathcal{C}\})$ ▷ Eqn. 3
  $\mathcal{I} \leftarrow \textsc{SelectSpans}(\boldsymbol{X}_\mathcal{C}, \{d_\mathcal{C}\}, W, B)$
  for each segment $I \in \mathcal{I}$ **do**
    $\mathbf{z} \leftarrow \textsc{TokenIDs}(I)$ ▷ I: token positions in the sequence
    $\mathcal{A} \leftarrow \textsc{AssignAddress}(\{\boldsymbol{q}_\mathcal{C}\}, \{d_\mathcal{C}\}, I)$ ▷ Eqn. 4
    $\mathcal{NB}.\texttt{append}(\mathbf{z}, \mathcal{A})$
  end for
end for

2. Address *Process the question for retrieval*

$\mathcal{Z} \leftarrow [\,]$
if $|\mathcal{NB}| > 0$ **then**
  $\mathcal{S}_{\text{retrieval}} \leftarrow \texttt{copy}(\mathcal{S})$
  $_, _, \{\boldsymbol{u}\} \leftarrow \mathcal{M}_\phi.\texttt{forward}(\mathcal{S}_{\text{retrieval}}, \boldsymbol{Q})$ ▷ $\boldsymbol{u} \leftarrow \text{compact}(\boldsymbol{q_Q})$ omitted
  $V \leftarrow \textsc{Address}(\{\boldsymbol{u}\}, \mathcal{NB}.\texttt{addresses})$ ▷ Eqn. 5
  $\{\mathbf{z}_{\text{evidence}}\} \leftarrow \textsc{Top-R}(\mathcal{NB}, V)$
  $\mathcal{Z} \leftarrow \texttt{concat}(\{\mathbf{z}_{\text{evidence}}\}.\texttt{sorted()})$ ▷ $\{\mathbf{z}_{\text{evidence}}\}$ sorted by input order
  $\texttt{delete}(\mathcal{S}_{\text{retrieval}})$
end if

3. Read *Replay, then answer*

$\mathcal{S}_{\text{read}} \leftarrow \texttt{copy}(\mathcal{S})$
$\boldsymbol{R} \leftarrow \mathcal{Z}$
$\mathcal{S}_{\boldsymbol{R}} \leftarrow \mathcal{M}_\phi.\texttt{forward}(\mathcal{S}_{\text{read}}, \boldsymbol{R})$ ▷ Eqn. 6
$Y \leftarrow \mathcal{M}_\phi.\texttt{generate}(\mathcal{S}_{\boldsymbol{R}}, \boldsymbol{Q})$ ▷ Eqn. 6
return Y

then on the right. Each sentence is included only if the expanded span remains within the B-token budget. This process repeats until neither side can be added. Expansion follows sentence order and the length budget, without further salience comparisons. If the initial sentence alone exceeds B tokens, we retain a contiguous B-token portion starting at the selected window's start, adjusted to the nearest valid position that keeps the portion entirely within the sentence.

When writing two notes per chunk, we select the first note and exclude all sentences that overlap it. If the note contains only part of a long sentence, we exclude that entire sentence. This leaves at most two candidate regions, one on each side of the excluded text. We find the highest-scoring window across these regions and construct the second note using the same procedure. If no candidate remains, we write only one note. The two notes from a single write operation therefore do not overlap, although notes written at different chunk boundaries may overlap. For each note, we use the retained per-channel distinctiveness scores to select the query positions that form its addresses, as defined in Eqn. 4. Our implementation supports $W \in \{1, 2\}$. Any remaining tokens in a final incomplete chunk are processed by the backbone but produce no notes. A context shorter than C tokens therefore leaves the archive empty.

B.3 Native address construction

For each selected note, we first identify one representative token per recurrent channel using the native-key distinctiveness scores, as defined in Eqn. 4. We then compact the native query at that position. Thus, representative-token selection uses key distinctiveness, while the stored address consists of query features.

We implement compact as fixed coordinate selection. For each channel (ℓ, h), we sample an ordered set of d_ρ distinct coordinates from the native query dimension d_k, independently across channels, and keep this selection fixed across documents and questions. Let $P^{\ell,h} \in \{0,1\}^{d_\rho \times d_k}$ denote the corresponding coordinate-selection matrix. The note and question representations are

$$\boldsymbol{\rho}_i^{\ell,h} = \mathrm{compact}\left(\boldsymbol{q}^{\ell,h}_{t_i^{*,\ell,h}}\right) = P^{\ell,h}\boldsymbol{q}^{\ell,h}_{t_i^{*,\ell,h}}, \qquad \boldsymbol{u}_\tau^{\ell,h} = \mathrm{compact}\left(\boldsymbol{q}_\tau^{\ell,h}\right) = P^{\ell,h}\boldsymbol{q}_\tau^{\ell,h}. \tag{9}$$

Each row of $P^{\ell,h}$ selects one coordinate, with no coordinate repeated within a channel. The same mapping is used for stored notes and incoming questions; it introduces no trainable parameters. Our standard configuration uses $d_\rho = 16$ and a fixed set of randomly-generated coordinates. Projected note addresses are stored in `BF16` without normalisation. The question representations are also converted to `BF16`. During matching, both are cast to `FP32` and ℓ_2-normalised before computing cosine similarities. Compaction does not change which representative position is selected: that decision is made from the per-channel native-key distinctiveness scores before address compression.

B.4 Replay readout and optional adaptation

The retained inference state $\mathcal{S}_T$ includes recurrent states, convolution caches, and, for hybrid backbones, attention key-value caches and position information. Retrieval and answer generation use separate copies of this state. As shown in Algorithm. 1, the question is first processed on a temporary copy to select notes; this copy is then discarded. On the answer branch, the selected notes are concatenated in document order and replayed before the question is processed again. Replay uses the original token IDs and follows the backbone's configured attention policy. Position indices continue from the context boundary rather than returning to the notes' original positions. An empty archive produces zero replay tokens.

Optional adaptation adds low-rank updates $\Delta W = (\alpha/r)BA$ to selected backbone projections, while keeping pretrained weights frozen. Here r is the adapter rank and α controls its scale; B is initialised to zero. For GDN backbones, adapters modify the query, key, value, and output projections. For Mamba-2, they modify the x, B, and C projection branches and the output projection. Adapters are disabled during context processing and retrieval, and enabled during replay, question processing, and answer generation.

We train the adapters using the mean negative log-likelihood of the target answer $\widetilde{\boldsymbol{Y}}$, including its EOS token:

$$\mathcal{L} = -\frac{1}{|\widetilde{\boldsymbol{Y}}|}\sum_{t=1}^{|\widetilde{\boldsymbol{Y}}|}\log p_{\phi,\theta}\big(\widetilde{\boldsymbol{y}}_t \mid \mathcal{S}_{\mathrm{read}}, \boldsymbol{R}, \boldsymbol{Q}, \widetilde{\boldsymbol{y}}_{<t}\big), \tag{10}$$

where ϕ denotes the frozen backbone weights and θ the adapter parameters. Only answer tokens contribute to the loss. The context state and discrete note selection are treated as constants, while gradients propagate through replay and subsequent question answering to the adapters.

C Experimental Details

C.1 Backbones

We evaluate NoteBank on GDN, GDN-2, Mamba-2, Mamba-2-8B, and Qwen3.5-9B. For the three smaller recurrent backbones, we use `m-a-p/1.3B-100B-GatedDeltaNet-pure`, `LLM-OS-Models2/gdn2-1.3b-paper-matched`, and `state-spaces/mamba2-1.3b`, respectively. GDN and GDN-2 contain 24 and 18 recurrent layers, yielding 192 and 288 indexed layer-head channels. Mamba-2 contains 48 layers with one B/C group per layer, yielding 48 independent addressing channels. GDN-2 and Mamba-2 checkpoints are converted to the local inference implementations, using their corresponding tokenizers. For Mamba-2-8B, we use NVIDIA's `nvidia/mamba2-8b-3t-4k`, an 8B-parameter pure Mamba-2 base model pretrained on 3.5T tokens with a sequence length of 4K. Qwen3.5-9B (`Qwen/Qwen3.5-9B`) combines 24 Gated DeltaNet blocks with 8 Transformer self-attention blocks, giving a recurrent-to-attention ratio of 3:1. NoteBank uses native key and query features exclusively from the Gated DeltaNet blocks for writing and addressing, yielding 384 indexed layer-head channels. Replayed tokens pass through the entire hybrid backbone, including both block types, while the attention blocks retain their configured

attention policy and KV caching. Pretrained backbone weights remain frozen in all NoteBank experiments; adapted variants train only the additional adapters. The selected backbones in summary:

- GDN (1.3B): `m-a-p/1.3B-100B-GatedDeltaNet-pure`.
- GDN-2 (1.3B): `LLM-OS-Models2/gdn2-1.3b-paper-matched`.
- Mamba-2 (1.3B): `state-spaces/mamba2-1.3b`.
- Mamba-2 (8B, NVIDIA): `nvidia/mamba2-8b-3t-4k`.
- Qwen3.5-9B: `Qwen/Qwen3.5-9B`.

C.2 Benchmarks and evaluation protocols

RULER-NIAH (Hsieh et al., 2024). We evaluate six retrieval tasks: `niah_single_1`, `niah_single_2`, `niah_single_3`, `niah_multikey_1`, `niah_multiquery`, and `niah_multivalue`. We use 100 fixed examples per task at each target length: 4K, 16K, 64K, 256K, and 1M tokens, where K denotes 2^{10}. We apply the benchmark's task-specific scoring and report the unweighted mean across the six tasks at each length.

LongBench-v1 (Bai et al., 2024). We evaluate 12 subsets covering question answering, summarisation, few-shot tasks, and code completion: `narrativeqa`, `qasper`, `multifieldqa_en`, `hotpotqa`, `2wikimqa`, `musique`, `gov_report`, `qmsum`, `multi_news`, `trec`, `triviaqa`, `samsum`. We use 30 fixed examples per subset (360 samples in total), shared across methods. The recurrent base models receive the native task prompts without additional chat formatting. We use the task-specific metrics and report their unweighted macro-average on a 0-100 scale.

MemoryAgentBench (Hu et al., 2026). We evaluate Mamba-2-8B on 11 task configurations: FactConsolidation-sh and FactConsolidation-mh at 6K, 32K, and 262K; EventQA at 64K, 128K, and full length; and RULER QA1 and QA2. Each FactConsolidation and RULER configuration contains 100 evaluation questions, while each EventQA configuration contains 500, yielding 2,100 questions in total. The measured input lengths range from approximately 7K to 746K tokens. We compare plain inference, a clean-start baseline, and NoteBank with retrieval budgets $R = 1$ and $R = 4$. We report accuracy (%) for each configuration and their unweighted macro-average.

LongBench-v2 (Bai et al., 2025). For Qwen3.5-9B, we evaluate 375 full-input examples from `Single-Document QA`, `Multi-Document QA`, `Long-dialogue History Understanding`, and `User guide QA`. Examples are retained when the complete prompt fits within the position budget after reserving space for replay and generation; inputs are not truncated. We use the official zero-shot prompt and answer parser, with the model's chat template and thinking disabled. Decoding is greedy with a maximum of 32,768 generated tokens, and the primary metric is accuracy over examples. This evaluation covers the specified subset rather than the full benchmark.

C.3 NoteBank and baseline configurations

For the short-note GDN, GDN-2, and Mamba-2 experiments, we use up to $W = 2$ notes per completed chunk, a note budget of $B = 64$ tokens, a 48-token salience window, and a 32-token rolling tail. The chunk size is $C = 512$ for GDN and GDN-2 and $C = 256$ for Mamba-2. Each note stores one $d_\rho = 16$ dimensional address per channel, with random coordinate selection. We evaluate retrieval budgets $R \in \{1, 2, 4\}$. The plain baseline processes the original prompt without note replay; JRT-Prompt processes the context and question twice before generating an answer. Generation is greedy, with the same task-specific output limits and stopping rules across compared methods.

On very long inputs, we observe substantial recurrent-state magnitude explosion and degraded readout in Mamba-2 before any replay is performed. *This instability already occurs in unmodified Mamba-2, independently of NoteBank or evidence replay*. Related work identifies state-distribution and dynamical mismatches as sources of length-generalisation failures (Ruiz & Gu, 2025; Lu et al., 2025). In our long-context setting, reliable readout therefore requires rescaling the recurrent state at the readout boundary $\boldsymbol{S}^{\ell,h} \leftarrow \lambda \boldsymbol{S}^{\ell,h}$ with a factor $\lambda \in [0, 1]$, where $\lambda = 1$ leaves the state unchanged and $\lambda = 0.$ resets it. Small values of λ often yield better performance on our longest-context tasks, both with and without NoteBank.

For NoteBank, we use $\lambda = 0$. immediately before replay while retaining convolution caches and position information. Document ingestion, note construction, and question-conditioned retrieval remain unchanged. Decoupling archival memory from recurrent computation makes this intervention possible: historical evidence remains accessible through retrieved text even after the accumulated recurrent state is discarded. Resetting alone also improves over unmodified Mamba-2 (Appendix D.1), suggesting that the accumulated state can impair subsequent readout in these settings.

We retain unmodified Mamba-2 as the main-text reference to report its out-of-the-box performance. To distinguish the effects of state rescaling from those of memory augmentation, this appendix additionally reports Mamba-2 baselines with the corresponding rescaling applied before question processing. NoteBank retains its advantage under these matched controls, showing that its gains are not explained by state rescaling alone.

C.4 Adaptation setup

For each backbone and retrieval budget, we fine-tune the decoder on the same 2,400-example corpus using *fewer than 9.3M tokens*, then evaluate the resulting adapter across all 12 LongBench tasks without task-specific fine-tuning. We construct this adaptation corpus from Natural Questions-derived short-answer examples in `emozilla/Long-Data-Collections-Fine-Tune`. Training contexts are capped at 16,000 characters, with gold-document identifier trailers removed and answers restricted to 1-24 words. Preserving the supplied data splits, the resulting pool contains 5,855 examples: 4,157 answerable and 1,698 natural no-support examples containing neither the answer string nor a gold-passage match. For both top-1 and top-2 adaptation, we use the same 2,400 examples without replacement after shuffling once with seed 0, comprising 1,722 answerable and 678 no-support examples. The total fine-tuning budget is below 9.3M tokens. No-support examples retain their original gold answers; supervised targets contain at most 47 answer tokens followed by EOS. A separate set of 354 no-support validation examples is used only for post-training diagnostics.

We use LoRA rank $r = 8$, scale $\alpha = 16$, and zero-initialised output factors at zero-indexed layers $\{11, 15, 18, 22\}$ for GDN, $\{8, 11, 13, 16\}$ for GDN-2, and $\{23, 31, 37, 45\}$ for Mamba-2-1.3B. Training uses 600 AdamW updates with four examples accumulated per update, learning rate 5×10^{-4}, weight decay 0.01, gradient clipping at 1.0; evaluation uses the final checkpoint. Notes and routing are recomputed online with $C = 512$, $W = 2$, and $R \in \{1, 2\}$. Adapter activation and the answer-only training objective follow App. B.

C.5 Implementation and efficiency measurement

Efficiency measurements use an NVIDIA H100 80GB GPU, `BF16` precision, and batch size 1, with one benchmark process per GPU. For the GDN comparison, NoteBank uses $C = 512$, $W = 2$, $B = 64$, and $R = 2$. At each target length of 4K, 64K, and 1M, we measure three fixed RULER inputs: example 0 from `niah_single_2`, `niah_multiquery`, and `niah_multivalue`. Each input-method pair receives one warm-up run and three timed runs, with method order alternated. We report the mean of the per-input medians.

Timing covers complete prompt processing and generation of exactly 32 new tokens, with EOS and stop-string termination disabled. For NoteBank, this includes archive construction, question probing, routing, and replay; the archive is rebuilt for every run. Model loading, initial prompt tokenisation, warm-up, and result serialisation are excluded. Any additional tokenisation performed inside the NoteBank inference call remains included.

GPU memory is measured using `torch.cuda.max_memory_allocated()` and reported as an absolute peak including resident model weights and temporary allocations. Analytical persistent-storage estimates and host-side storage are reported separately. These fixed-output cost measurements are independent of the task-specific generation and scoring protocols used for quality evaluation.

For the Transformer comparisons, Full, SW4K, and InfLLM use the same checkpoint as in the main text, `m-a-p/transformer_1.3B_baseline`, with approximately 1.148B unique parameters, 24 layers, hidden width 2048, and 32 query/8 KV heads. Full uses FLA's causal FlashAttention and retains all historical KV across 512-token prefill chunks, whereas SW4K uses the same chunk size with a 4096-token causal window including the current token and no additional sinks. Both use

unscaled RoPE with $\theta = 10{,}000$ and monotonically increasing absolute positions; SW4K computes rotations on demand without altering this formula. InfLLM also uses 512-token chunks, processes the final prompt token separately, retains 128 initial tokens, and applies its official local/global positional treatment with the same RoPE base and unit distance scale. We apply no additional training or RoPE scaling: 1M inputs exceed the checkpoint's configured 4096-position range, although this configuration alone does not establish its original training sequence length.

C.6 Added FLOPs Accounting

We estimate the additional inference FLOPs of NB-TF relative to the same backbone without NoteBank, counting multiply-adds as two FLOPs. Accounting covers context ingestion through answer generation, using recorded input, archive, and replay lengths. Both methods use the corresponding Plain answer length $|\boldsymbol{Y}|$ to isolate method overhead. Let $J = |\mathcal{L} \times \mathcal{H}|$ and $|\boldsymbol{R}| \le RB$ be the replay length.

Backbone cost. Let f_ϕ denote backbone FLOPs per token, excluding the language-model head, whose cost is $h_\phi = 2DV$ for hidden width D and vocabulary size V. We sum $2d_{\text{in}}d_{\text{out}}$ over projection and MLP matrices using checkpoint shapes, then add convolution and recurrent-equivalent costs f_{conv} and f_{rec}. For GDN,

$$\begin{aligned} f_{\text{conv}} &= 2c_{\text{conv}}J(2d_k + d_v), \\ f_{\text{rec}} &= J(7d_kd_v + d_k + 2d_v), \end{aligned} \tag{11}$$

where $c_{\text{conv}} = 4$. The recurrent term covers state decay, key and query readouts, the rank-one update, and vector scaling. Our GDN configuration gives $f_\phi \approx 2.759 \times 10^9$ FLOPs/token and $h_\phi \approx 1.311 \times 10^8$ FLOPs/prediction.

NoteBank overhead. For a purely recurrent backbone,

$$F_{\text{Plain}} = (T + |\boldsymbol{Q}| + |\boldsymbol{Y}| - 1)f_\phi + |\boldsymbol{Y}|h_\phi. \tag{12}$$

The final input position predicts the first answer token, requiring only $|\boldsymbol{Y}| - 1$ subsequent backbone steps. With a nonempty archive,

$$\begin{aligned} \Delta F &\equiv F_{\text{NB-TF}} - F_{\text{Plain}} \\ &= F_{\text{write}} + F_{\text{route}} + (|\boldsymbol{Q}| + |\boldsymbol{R}|)f_\phi. \end{aligned} \tag{13}$$

The additional backbone passes are the question match and replay; ordinary question processing and answer generation cancel. An empty archive incurs neither match nor routing or replay.

For the evaluated writer ($W = 2$, with a 32-token tail),

$$\begin{aligned} F_{\text{write}} \approx\ & 6TJd_k + 4\lceil T/C \rceil Jd_k + 2TJ \\ & + 6(C + 32)\lfloor T/C \rfloor. \end{aligned} \tag{14}$$

The first three terms estimate key normalisation, chunk-mean construction, distinctiveness scoring, and channel aggregation; the last conservatively bounds window-selection arithmetic. D addressing reuses these scores to select representative positions and gathers their compact query features, requiring no additional backbone pass or dense projection.

Matching every question token against all N_T notes costs

$$F_{\text{route}} \approx 2|\boldsymbol{Q}|N_TJd_\rho + 3(|\boldsymbol{Q}| + N_T)Jd_\rho, \tag{15}$$

including dot products and address normalisation.

Aggregation and interpretation. We calculate costs per example and report

$$\text{Added FLOPsratio} = 100\% \times \frac{\sum_e \Delta F_e}{\sum_e F_{\text{Plain},e}}. \tag{16}$$

For GDN with top-2 replay on 600 RULER examples at the 1M target length, mean Plain and additional costs are approximately 2891.4 and 0.722 TFLOPs, respectively, giving 0.025% added FLOPs. Values are rounded after per-example accounting.

These estimates cover principal arithmetic, excluding most backbone pointwise operations, comparisons, sorting, indexing, data movement, and evaluator-only work. They use recurrent-equivalent costs rather than exact chunk-kernel execution counts and do not measure latency. The accounting excludes LoRA; hybrid backbones also require attention costs induced by replay, which do not cancel as above.

C.7 Persistent Memory Accounting

We compare NoteBank's persistent state-and-archive payload with a full `BF16` MHA KV cache matched in hidden width D and layer count $|\mathcal{L}|$. For input length $P = T + |\boldsymbol{Q}|$, the reference occupies

$$M_{\mathrm{MHA}} = 4|\mathcal{L}|DP \quad \text{bytes}, \tag{17}$$

counting keys and values at two bytes per element, without KV-head sharing.

The GDN used in our experiments has 24 layers and 8 heads per layer, with $J = 192$ and $d_k = d_v = 256$, giving $D = 2048$. NoteBank uses $C = 512$, $W = 2$, $B = 64$, and $d_\rho = 16$. Across 600 RULER examples at the 1M target length, the mean input length, note count, and archived-token count $U = \sum_{i=1}^{N_T} |\mathbf{z}_i|$ are

$$\bar{P} \approx 1{,}048{,}019, \qquad \overline{N_T} \approx 4092.3, \qquad \bar{U} \approx 232{,}514. \tag{18}$$

We use these recorded counts rather than assuming full-length notes or exactly 2^{20} input tokens.

Charging addresses as BF16, token IDs as compact int32, and metadata as three uint64 fields per note (`chunk`, `start`, `end`), the archive payload is

$$\mathrm{Storage}_{\mathcal{NB}} = 2N_T J d_\rho + 4U + 24N_T \quad \text{bytes}. \tag{19}$$

Including `FP32` recurrent states, `BF16` Q/K/V convolution caches, and resident helper tensors gives

$$\begin{aligned} M_{\mathrm{NB}} = \mathrm{Storage}_{\mathcal{NB}} &+ 4Jd_k d_v \\ &+ 2c_{\mathrm{conv}} J(2d_k + d_v) + M_{\mathrm{helper}}. \end{aligned} \tag{20}$$

Here $c_{\mathrm{conv}} = 4$ and $M_{\mathrm{helper}} \approx 0.402$ MiB include projection indices and allocated auxiliary tensors, including those unused by routing. Using $1\,\mathrm{MiB} = 2^{20}$ bytes, recurrent states, convolution caches, and the archive occupy 48, 1.125, and 24.959 MiB on average. Thus,

$$\frac{\overline{M_{\mathrm{NB}}}}{\overline{M_{\mathrm{MHA}}}} \approx \frac{74.49}{196{,}503.5} \approx \frac{1}{2638}. \tag{21}$$

This is a ratio of aggregate payloads, with rounding applied after accounting.

The comparison is taken at the start of answer generation and includes the entire archive. It excludes model weights, transient workspace, and encoding/replay cache copies. CPU text and metadata are charged at compact representation sizes, excluding Python-object overhead; these figures therefore describe payload, not measured CPU heap usage or peak GPU allocation. The ratio applies to this GDN configuration at 1M.

D Additional Results

D.1 Complementary results

Per-task retrieval results. Section 4.1 reports accuracy averaged over six RULER tasks. Here, we provide the per-task breakdown for GDN, GDN-2, and Mamba-2 across context lengths from 4K to 1M, using the same checkpoints, methods (w/ and w/o NB-TF), and 100 examples per task. The breakdown shows that NoteBank's gains vary across task types: single-needle retrieval is generally stronger, while multi-query and multi-value tasks remain more challenging under a limited replay budget. NB-TF sustains substantial retrieval accuracy at long contexts where Plain and JRT largely fail. However, recovery is task-dependent: strong single-needle retrieval does not uniformly translate into successful multi-query or multi-value retrieval. The per-task breakdown also reveals greater variability for Mamba-2, which ties with only Mamba-2, showing that a shared memory interface does not eliminate backbone-specific limitations.

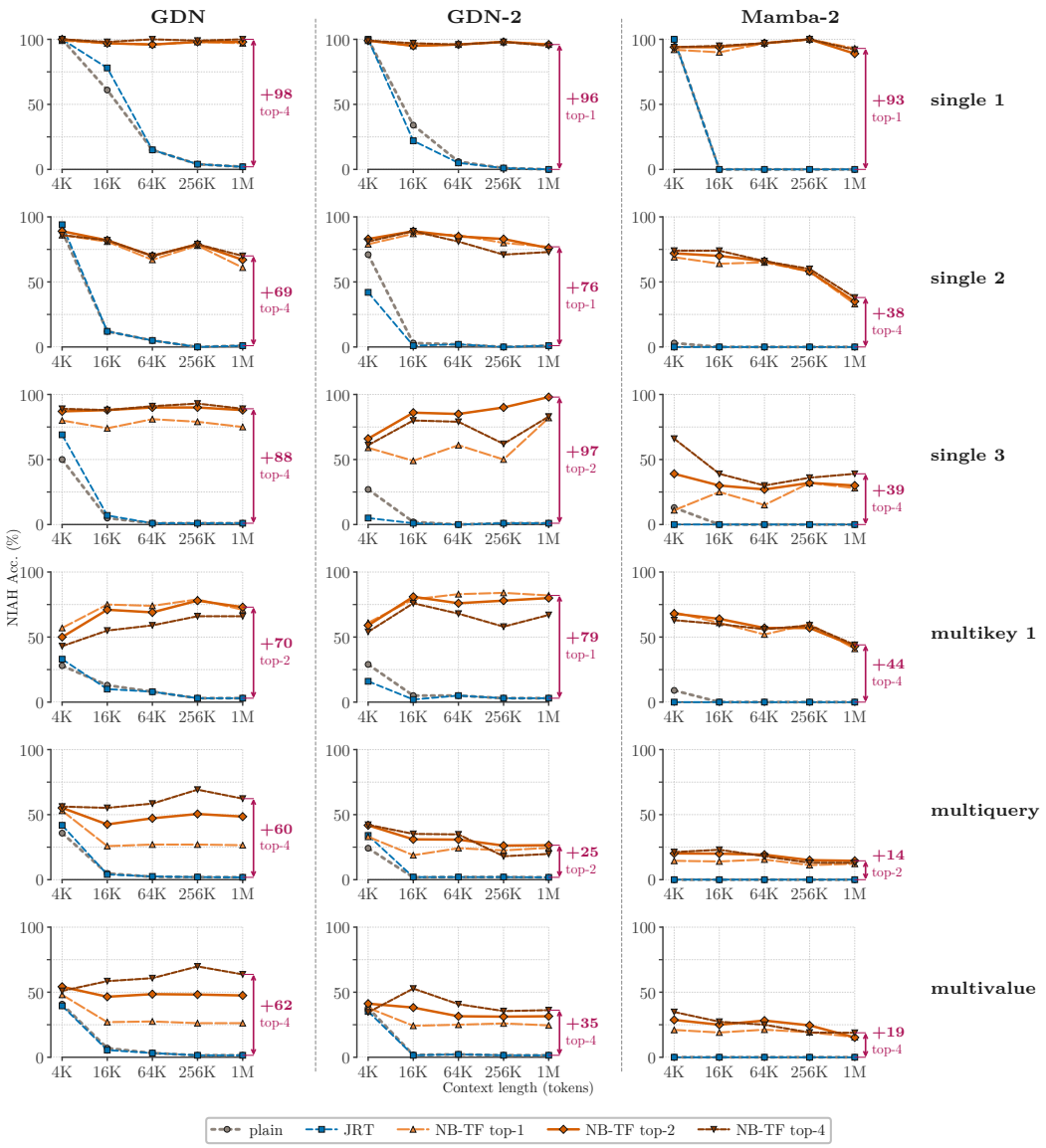


Figure 6: **Training-free long-range retrieval across architectures and context lengths:** full results on RULER 6 NIAH tasks.

Comparison with pretrained state-caching methods. Table 5 extend Sec. 4.2 with published results for Memory Caching (MC; Titans+SSC) (Behrouz et al., 2026) and MARCH (Zhang et al., 2026). Both incorporate state caching during pretraining and use 16K training sequences for their LongBench results; MARCH is pretrained on 50B tokens. These results provide contextual references, given differences in model sizes and pretraining protocols. Using an off-the-shelf GDN checkpoint pretrained on short contexts (`m-a-p/1.3B-100B-GatedDeltaNet-pure`, trained on $2K$), NB-TF achieves 16.07 overall, exceeding MARCH (14.87) and approaching MC (16.50), with *no additional training*. NB-LoRA further reaches 23.03 using fewer than 9.3M adaptation tokens. Beyond LongBench, GDN with NB-TF retains approximately 70% mean accuracy across six RULER tasks at 1M tokens (Sec. 4.1), without context-extension training. Together, these results demonstrate that effective long-range memory can be added to existing checkpoints without pretraining a specialised memory architecture.

Backbone	Method	Single-doc QA	Multi-doc QA	Summarisation	Few-shot	Overall
Transformer†	Plain	13.40	18.97	8.43	23.40	16.05
Titans†	SSC	12.80	18.07	7.63	27.50	16.50
GDN‡	MARCH	8.87	7.90	13.23	29.47	14.87
GDN	Plain	6.88	7.02	13.99	34.08	15.49
	JRT	7.72	6.63	**14.79**	33.56	15.67
	NB-TF	6.66	7.33	13.52	36.76	16.07
	NB-LoRA	**19.06**	**20.48**	11.37	**41.22**	**23.03**
Mamba-2	Plain	2.22	2.10	11.27	11.56	6.79
	JRT	2.39	1.89	9.21	8.34	5.12
	NB-TF	5.98	5.36	**16.51**	25.88	13.43
	NB-LoRA	**17.84**	**15.54**	7.46	**27.52**	**17.10**

Table 5: **From retrieval to real-world long-context tasks.** Scores are macro-averaged over evaluated LongBench-v1 tasks, within each category and overall. NB-TF requires no additional training; NB-LoRA adapts the read stage. †Reference results aggregated from Behrouz et al. (2026); ‡Reference results aggregated from Zhang et al. (2026). Both † and ‡ require pretraining on 16K-context length data. Bold marks the best score within each recurrent backbone.

Backbone	Method	FactCon. Single-hop	FactCons. Multi-hop	EventQA	RULER QA	Overall
Mamba-2-8B	Plain	0.0	0.0	0.0	0.0	0.0
	Clean start	14.0	2.0	17.7	16.0	13.0
	NB-TF ($R = 1$)	19.0	**3.0**	**17.9**	16.5	14.5
	NB-TF ($R = 4$)	**22.0**	2.5	16.4	**17.0**	**14.7**
Qwen3.5-9B	Plain	42.5	**5.5**	90.2	67.5	55.7
	NB-TF(top-1)	42.5	**5.5**	90.2	67.5	55.7
	NB-TF(top-4)	**45.0**	**5.5**	**90.7**	**69.0**	**56.8**

Table 6: **MemoryAgentBench results by task family and replay budget.** Accuracy (%) for Mamba-2-8B and Qwen3.5-9B, using matched examples within each backbone. RULER QA differs from the RULER-NIAH tasks reported in this paper. Overall reports the macro-average over evaluated tasks. For Mamba-2, Clean start resets the inference state without replay, while NB-TF additionally replays retrieved evidence. Bold marks the best score within each backbone, including ties.

Task-family results and replay budgets on MemoryAgentBench. Table 6 expands Section 4.3 with task-family results at $R = 1$ and $R = 4$. For Qwen3.5-9B, top-1 matches Plain, while top-4 increases overall accuracy from 55.7 to 56.8. Larger replay budgets are not uniformly beneficial: for Mamba-2-8B, increasing R improves single-hop FactConsolidation from 19.0 to 22.0 but reduces EventQA accuracy from 17.9 to 16.4.

State reset and evidence replay in Mamba-2. As discussed in App. C.3, Mamba-2 suffers severely from long-context state explosion, and by decoupling memorisation and recurrent computation, NoteBank facilitate a bypass: resetting the recurrent computation when performing QA and answering questions using only the retrieved evidence. Table 6, shows that clean start raises Mamba-2-8B's overall accuracy from 0.0 to 13.0, showing that the accumulated inference state substantially impairs readout. NB-TF further improves accuracy to 14.5 with top-1 and 14.7 with top-4 replay. The largest gain over Clean start occurs on single-hop FactConsolidation, from 14.0 to 22.0. Thus, resetting accounts for much of the recovery, while retrieved evidence provides additional, task-dependent benefits.

Efficiency breakdown. Figure 7 decomposes the additional analytical FLOPs of NB-TF into writing/indexing, question encoding, routing, and replay. Writing and indexing grow linearly with context length but reuse native features without another backbone pass. Question encoding and replay remain approximately constant for fixed question length and replay budget. Routing grows with the

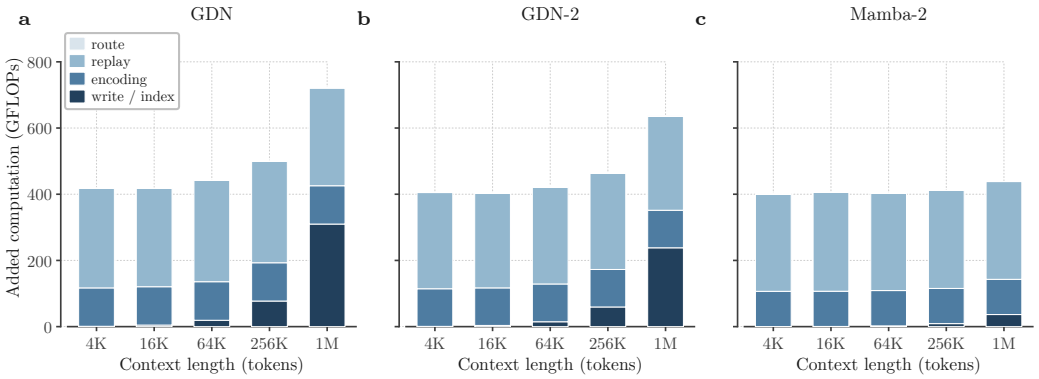


Figure 7: **Breakdown of NoteBank's computation overhead.** Added analytical FLOPs for NB-TF with top-2 replay relative to Plain on RULER, using matched output lengths. Stacked bars separate writing/indexing, question encoding, routing, and replay for GDN, GDN-2, and Mamba-2. Annotations report routing costs at 1M tokens in GFLOPs.

	Gated DeltaNet		Transformer		
Metric	Plain	**+ NB-TF**	Full	+ SW4K	+ InfLLM
RULER @ 1M ↑	1.71	**70.33**	0	1.17	0
GPU peak (GiB) ↓	**2.93**	3.27	55.18	5.41	7.30
CPU archive (MiB) ↓	0	0.98	0	0	48930.00

Table 7: **Retrieval quality vs. measured storage at 1M tokens.** RULER averages six NIAH tasks. GPU peak includes model weights. CPU archive reports compact token IDs and span-location records for NoteBank, and counted host KV for InfLLM.

archive size, yet matching compact addresses costs only 1.09, 1.72, and 0.50 GFLOPs at 1M tokens for GDN, GDN-2, and Mamba-2, respectively. Thus, operations that scan the context or archive remain lightweight, while additional backbone computation is confined to the question and retrieved evidence. Together, these costs add only 0.0171%–0.0250% to full-model analytical FLOPs at 1M tokens. These arithmetic estimates do not imply an equivalent latency overhead.

Practical runtime and memory discussion We measure latency and peak GPU memory on an H100 with batch size 1 and 32 generated tokens. Reference systems share a 1.3B Transformer backbone, using full attention, SW4K, or the official InfLLM implementation with default settings (Xiao et al., 2024). Detailed configurations appear in App. C.5. Figure 4c shows approximately linear latency scaling for GDN with and without NoteBank. Despite its negligible additional FLOPs, NoteBank incurs 10.6%–13.2% measured latency overhead across evaluated lengths. At 1M tokens, latency increases from 125.11 to 140.64 s, while peak GPU memory, including model weights, increases from 2.93 to 3.27 GiB (Table 7). The reference systems illustrate the broader quality-cost trade-off: full attention requires 55.18 GiB of peak GPU memory; InfLLM, configured with a 4K local attention window, reduces this to 7.30 GiB but stores 47.78 GiB of historical keys and values in CPU memory. SW4K is faster at 61.38 s, but scores only 1.17 on RULER, compared with NoteBank's 70.33. These comparisons characterise the evaluated systems; the paired GDN results isolate NoteBank's incremental cost, showing substantial retrieval gains with modest runtime and peak-memory overhead.

Retrieval: quality vs cost. Native routing improves RULER scores over input embeddings by 29.0 points on GDN and 16.8 points on Mamba-2, with measured match-and-vote time of only 26 ms at 1M. BM25 achieves the highest RULER scores, while LongBench favours different signals across backbones, indicating that native features are not uniformly optimal. These results support native routing as an effective, low-overhead default without an auxiliary retriever.

Routing Signal	Auxiliary Retriever	Time @ 1M (ms)	RULER @ 1M		LongBench	
			GDN	Mamba-2	GDN	Mamba-2
Random	`None`	–	1.7	0.0	21.96	13.83
Input embedding	`None`	7,600	30.8	20.8	22.40	**16.70**
Native (NB)	`None`	26	**59.8**	**37.6**	**23.03**	14.25
BM25	`Lexical index`	87	**64.0**	**52.7**	23.50	**18.66**
BGE	`External encoder`	2,630	45.4	34.0	**23.77**	17.24

Table 8: **Routing quality and computational cost.** Scores average six RULER tasks at 1M tokens and 12 LongBench tasks, with LoRA readers fixed across routers within each backbone and benchmark. Bold marks the best score within each auxiliary-retriever group. Native routing matches backbone features and aggregates channel votes; input embeddings are mean-pooled. Reported times use a 4092-note bank at 1M: Native includes question probing and voting, BM25 runs on CPU, and BGE reports full-bank encoding. Timing scopes differ and do not constitute a matched query-latency comparison; no Random timing is available. `None` indicates no auxiliary retriever, not zero computational cost.

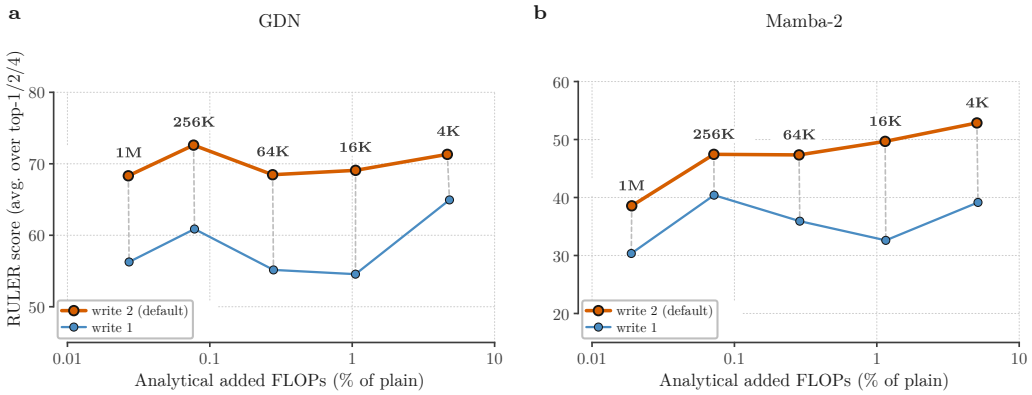


Figure 8: **Effect of the per-chunk write budget.** Six-task mean RULER accuracy versus analytical added FLOPs for NB-TF with $W \in \{1, 2\}$ on GDN and Mamba-2, averaged over replay budgets $R \in \{1, 2, 4\}$. Labels indicate context lengths. Writing two notes consistently improves accuracy with little additional arithmetic cost.

D.2 Ablation Study

We study the per-chunk write budget W, replay budget R, span budget B, and chunk size C on GDN and Mamba-2. Training-free ablations report mean accuracy across six RULER tasks from 4K to 1M tokens, together with analytical added FLOPs where applicable. We additionally examine the interaction between R and B with NB-LoRA on twelve LongBench tasks.

Write budget W. We test whether retaining more notes per chunk improves retrieval by comparing $W = 1$ and $W = 2$, averaging over $R \in \{1, 2, 4\}$ (Figure 8). Writing two notes consistently improves accuracy across both backbones and all evaluated context lengths. The gain comes with little change in analytical added FLOPs, since native feature extraction and distinctiveness scoring are shared across notes. This supports $W = 2$ as an effective way to improve archive coverage, at the cost of storing more notes.

Replay budget R. We vary $R \in \{1, 2, 4\}$ to assess the benefit and computational cost of replaying additional evidence (Figure 9). At 1M tokens, increasing R from 1 to 4 raises GDN accuracy from 59.5% to 75.2% and Mamba-2 accuracy from 37.1% to 40.8%. The benefit is not uniform: GDN peaks at $R = 2$ at 4K, and Mamba-2 shows small or negligible gains in some long-context settings. Larger budgets increase replay computation, but its relative cost decreases as context in-

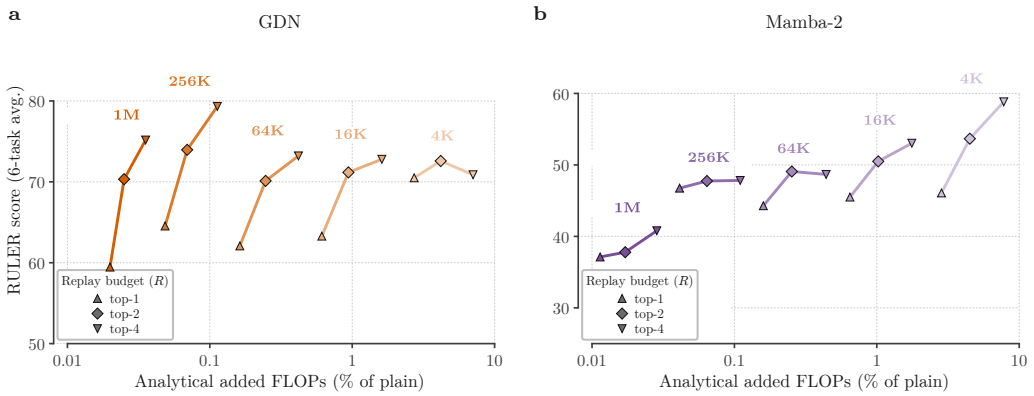


Figure 9: **Effect of the replay budget on retrieval and computation.** Six-task mean RULER accuracy versus analytical added FLOPs for NB-TF with $R \in \{1, 2, 4\}$ on GDN and Mamba-2. Labels indicate context lengths; connected markers compare replay budgets at the same length. Additional replay generally improves accuracy, with gains depending on the backbone and context length.

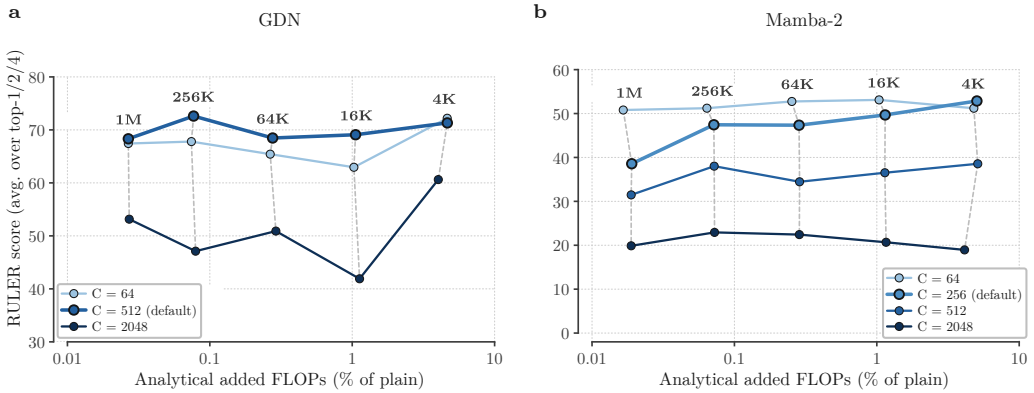


Figure 10: **Effect of chunk size on retrieval and computation.** Six-task mean RULER accuracy versus analytical added FLOPs for NB-TF with different chunk sizes C, averaged over replay budgets $R \in \{1, 2, 4\}$. Labels indicate context lengths. Smaller chunks particularly benefit Mamba-2 at long contexts, while very large chunks degrade retrieval for both backbones.

gestion accounts for more of the full-model FLOPs. Thus, additional replay can improve retrieval, with diminishing returns that depend on the backbone and context length.

Chunk size C. We vary C from 64 to 2048 to examine how writing granularity affects retrieval, averaging over $R \in \{1, 2, 4\}$ (Figure 10). GDN's default $C = 512$ matches or exceeds $C = 64$ across the evaluated lengths, whereas $C = 2048$ substantially reduces accuracy. Mamba-2 benefits more from frequent writing: $C = 64$ outperforms its default $C = 256$ at long contexts, while $C = 512$ and 2048 perform worse. The FLOPs differences are small, although smaller chunks permit denser archives and therefore increase storage requirements. These results show that useful writing granularity depends on the backbone, and that very coarse chunks can discard opportunities to preserve evidence.

Span budget B. We jointly vary $B \in \{64, 128, 256\}$ and $R \in \{1, 2, 4\}$ to distinguish longer notes from replaying more notes (Figure 11). For GDN, $B = 64$ performs best with top-1 replay at every evaluated length, while $B = 128$ often helps with larger replay budgets at long contexts. For Mamba-2, $B = 256$ consistently underperforms both smaller span budgets at matched replay budgets. At 1M, the best settings in this grid use $(B, R) = (128, 4)$, reaching 76.8% for GDN and

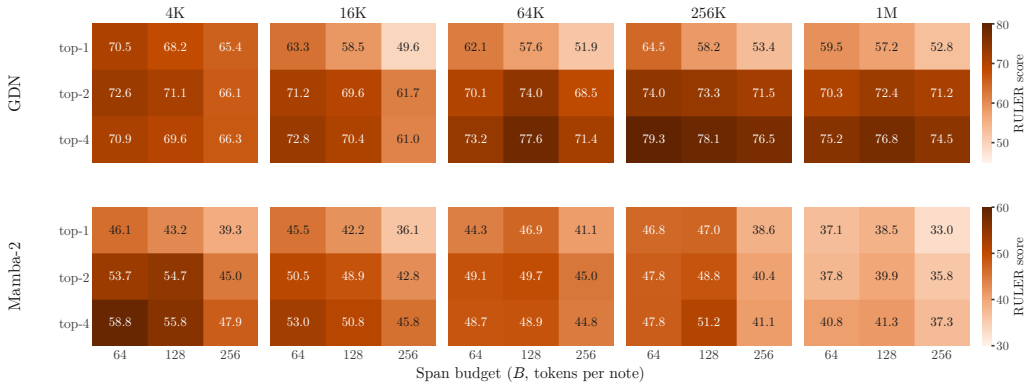


Figure 11: **Joint effects of span and replay budgets.** Six-task mean RULER accuracy (%) for NB-TF on GDN (top) and Mamba-2 (bottom), at context lengths from 4K to 1M. Each heatmap varies the span budget $B \in \{64, 128, 256\}$ across columns and replay budget $R \in \{1, 2, 4\}$ across rows. Longer spans do not consistently improve accuracy; their benefit depends on the replay budget and backbone.

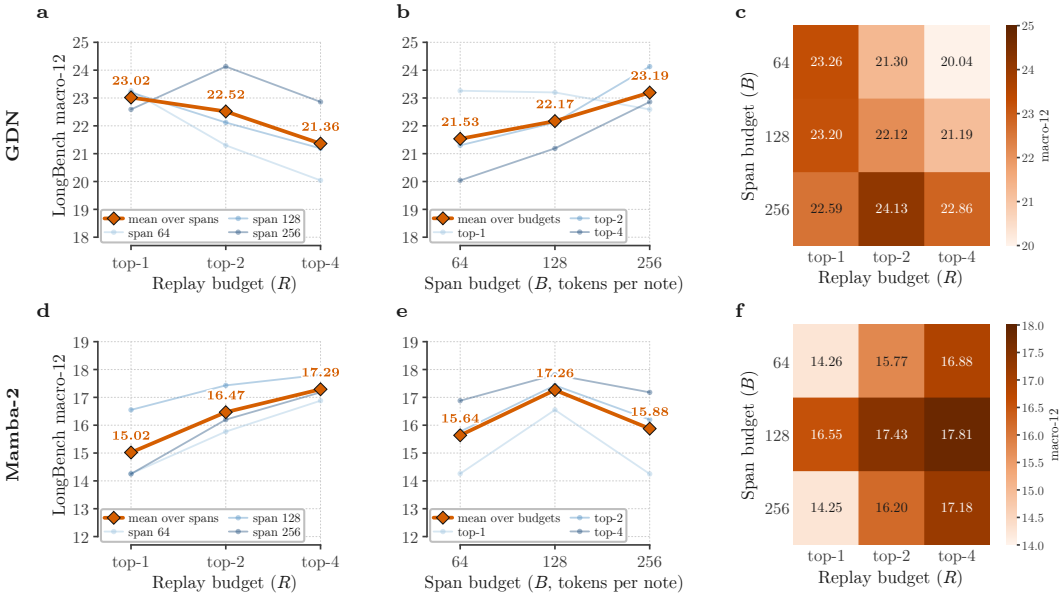


Figure 12: **Replay–span interaction with read adaptation.** LongBench macro-average scores over twelve evaluated tasks for NB-LoRA on GDN (top) and Mamba-2 (bottom). Left: varying replay budget R, with averages over span budgets. Middle: varying span budget B, with averages over replay budgets. Right: scores for each (B, R) configuration. The best replay and span budgets depend on the backbone.

41.3% for Mamba-2. Larger text budgets therefore do not automatically improve retrieval; span length and evidence coverage must be considered jointly.

Interaction: replay $\times$ span budget. We evaluate the same R-B grid with NB-LoRA on LongBench, reporting marginal averages and the full configuration heatmaps (Figure 12). Averaged over span budgets, increasing R from 1 to 4 reduces GDN's score from 23.02 to 21.36. However, the heatmaps reveal an interaction hidden by these averages: GDN performs best at $(B, R) = (256, 2)$ with a score of 24.13, while Mamba-2 peaks at 17.81 with $(B, R) = (128, 4)$. For Mamba-2,

$B = 128$ is also the best span budget at every replay budget. Read adaptation therefore does not remove sensitivity to evidence quantity and granularity, and the best configuration remains backbone-dependent.