PUM-NET: DUAL-MEMORY RETRIEVAL FOR LONG-CONTEXT STATE SPACE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context language modeling requires access to details whose relevance may emerge thousands of tokens after they appear. Selective state space models make long-sequence computation efficient, but recurrent compression offers no explicit route to revisit information discarded from the state. The challenge is to retain access to input-specific evidence while enabling its use alongside related corpus knowledge. We propose PUM-Net, a dual-memory extension of Mamba that separates recurrent computation, causal context retention, and offline corpus reuse. Internal retrieval accesses completed input chunks, while external retrieval accesses corpus representations encoded offline. Bidirectional attention updates each retrieved set using candidates from the other source before pooling and gated integration. This design preserves Mamba's parallel scan and incorporates retrieved evidence without appending passages to the input. After fine-tuning with 4k-token windows, the corpus-enabled configuration achieves the lowest perplexity in all twelve evaluated domain–length settings through 32k. At 130M, internal retrieval recovers passkeys in 18 of 20 single-trial settings beyond the training length, compared with 10 for Mamba. The results support explicitly retained memory as a complement to recurrent compression for prediction and recall over long contexts.

1 INTRODUCTION

Long-context language modeling requires access to information distributed throughout the input. A prediction may depend on an earlier definition, variable binding, or example that contributes little to the intervening text. Full attention provides direct access to preceding tokens, with a number of token-pair interactions that grows quadratically with sequence length (Vaswani et al., 2017; Tay et al., 2022). Selective state space models such as Mamba use input-dependent recurrent dynamics and a parallel scan (Gu & Dao, 2023), representing the preceding sequence through a fixed-width state.

Recurrent compression can retain information relevant to subsequent predictions, but provides no explicit mechanism for recovering a discarded detail. Long-context evaluations have identified recall failures beyond the training horizon (Chen et al., 2025; Yen et al., 2025). A content-addressed memory can supplement the recurrent state by retaining representations of earlier segments for subsequent retrieval.

The current sequence and an external corpus provide different forms of evidence. In a mathematical document, an earlier definition may determine the meaning of a symbol, while a related corpus passage supplies a derivation relevant to the current argument. Such corpus evidence can supplement information already represented in the model parameters. The two sources also differ in how they are constructed and updated: input memory grows with the observed prefix and requires causal masking, whereas corpus representations can be encoded offline and reused. PUM-Net maintains separate stores to accommodate these differences and conditions their retrieval on the current block output.

Within each augmented Mamba block, independent queries select values from the input and corpus stores. Bidirectional cross-attention operates on the two retrieved sets before pooling, allowing each candidate representation to depend on candidates from the other source. The pooled representations are then integrated through source-specific gates. Independent chunk encoding and the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

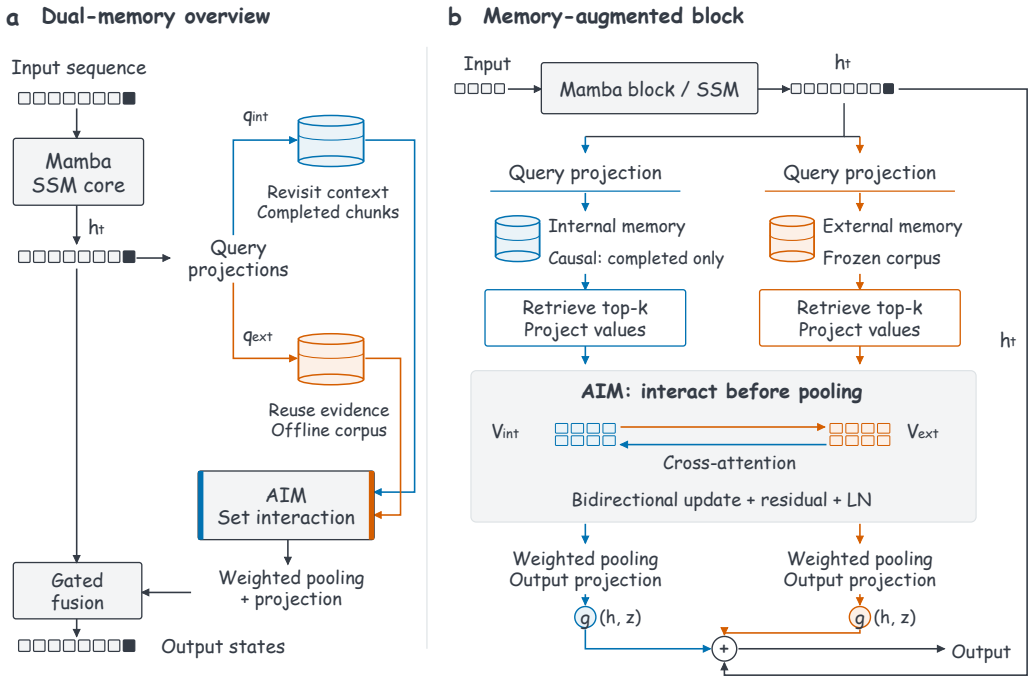


Figure 1: PUM-Net architecture. (a) Internal memory stores input-specific representations; external memory stores corpus representations encoded offline. (b) Each query retrieves up to eight values per source. AIM applies bidirectional cross-attention before pooling, output projection, and gated addition. Internal retrieval is restricted to completed chunks. The gray bypass carries h_t; gate labels abbreviate $g_t^s(h_t, z_t^s)$.

full-sequence scan are batched separately; retrieval requires neither passage concatenation nor a second full-sequence scan. Exact internal search nevertheless introduces a cost that increases with the number of stored chunks.

We evaluate long-range language modeling, passkey retrieval, task-specific question answering, and block-level computational cost. Internal retrieval improves passkey recovery relative to Mamba. Adding corpus memory reduces perplexity across the evaluated domains, with the magnitude depending on the dataset and context length. Figure 1 summarizes the architecture.

The principal contribution is a dual-memory interface for SSM blocks with distinct construction and availability rules for input and corpus representations. We define the causal retrieval mechanism, bidirectional interaction between retrieved sets, and gated integration into the block output. The evaluation compares internal-only and corpus-enabled configurations and quantifies their performance and computational costs.

2 Related Work

Recurrence and explicit memory. S4 and Mamba support efficient state-space sequence computation, and Mamba-2 relates selective SSMs to structured masked attention (Gu et al., 2022; Gu & Dao, 2023; Dao & Gu, 2024). Hybrid architectures combine SSM blocks with attention (Lieber et al., 2024). DeciMamba and MambaExtend address long-context behavior (Ben-Kish et al., 2025; Azizi et al., 2025); ReMamba selects and adapts hidden representations in a two-stage re-forward process (Yuan et al., 2024). Hierarchical Memory Mamba organizes working and long-term memory around paragraph-level representations (Wang et al., 2026). PUM-Net combines separately queried input and corpus stores within each augmented block, with one full-sequence scan and independently batched chunk encoding.

Representations retrieved from context. Transformer-XL, Compressive Transformer, and Recurrent Memory Transformer propagate or compress information beyond a local segment (Dai et al., 2019; Rae et al., 2020; Bulatov et al., 2022). Memorizing Transformers and LongMem retrieve earlier representations (Wu et al., 2022; Wang et al., 2023). Longformer, BigBird, and Native Sparse Attention restrict attention to selected long-range connections (Beltagy et al., 2020; Zaheer et al., 2020; Yuan et al., 2025). PUM-Net implements content-based retrieval within SSM blocks, with internal entries restricted to chunks completed before the next token is predicted.

Corpus retrieval and evidence interfaces. REALM and RAG condition language models on retrieved evidence (Guu et al., 2020; Lewis et al., 2020). RETRO encodes retrieved neighbors separately and integrates them through chunked cross-attention (Borgeaud et al., 2022), while kNN-LM queries a representation datastore at prediction time (Khandelwal et al., 2020). In PUM-Net, corpus values are encoded offline, input-specific values are constructed online, and both are projected into the receiving SSM block. Offline encoding reduces repeated value computation at the cost of storage, retrieval, and a fixed corpus representation.

Interaction between sources. Co-attention and gated cross-attention are established mechanisms for integrating representations (Lu et al., 2016; Alayrac et al., 2022). AIM applies bidirectional cross-attention to the two retrieved sets before pooling. Each candidate is therefore updated using candidates from the other source, after which query-dependent weights aggregate each set. This ordering integrates cross-source interactions into the SSM block while retaining separate retrieval and gating for the two memories.

3 PUM-NET

We describe one PUM block; the experiments replace selected Mamba blocks with this construction. Let $X = (x_1, \ldots, x_L)$ be the token sequence and $U = (u_1, \ldots, u_L)$ its input representations at this block. The Mamba transformation f_θ produces preliminary token outputs

$$H = f_\theta(U) = [h_1, \ldots, h_L], \qquad h_t \in \mathbb{R}^d. \tag{1}$$

Independent chunk encoding produces values for the internal store $\mathcal{M}_{\text{int}}$. These values and those from the external store $\mathcal{M}_{\text{ext}}$ augment the preliminary output h_t. Both stores use frozen text embeddings as retrieval keys. The value readout r_θ returns a terminal chunk representation, distinct from h_t; the two sources may have different value widths before projection.

3.1 TWO STATE MEMORIES

Internal memory. Let $I_j = \{(j-1)L_c + 1, \ldots, jL_c\}$ denote the interval of a complete chunk, with $L_c = 64$. The frozen key encoder E_k operates on token text X_{I_j}, whereas the block's chunk encoder and terminal readout operate on the block-input representations U_{I_j}:

$$k_j^{\text{int}} = E_k(X_{I_j}), \qquad s_j^{\text{int}} = r_\theta(U_{I_j}). \tag{2}$$

Complete chunks in a minibatch are encoded in one batched pass; a shorter suffix remains outside the store. For a next-token prediction made from position t, only chunks completed by that point are eligible:

$$\mathcal{A}_t = \{j : jL_c \le t\}. \tag{3}$$

The incomplete current chunk and all future chunks receive a $-\infty$ retrieval logit. This mask is applied during training and evaluation. When fewer than k chunks are available, retrieval uses all eligible entries and masks padding; when the set is empty, the internal update is hard-set to zero after projection and gating. When either retrieved set is empty, cross-attention is bypassed for that position and the nonempty set is pooled without interaction.

For example, no internal entry is available at $t = 63$. At $t = 64$, the first chunk becomes eligible for predicting token 65; the second chunk becomes eligible at $t = 128$. Thus chunk encoding can be batched while each prediction retains the same availability rule as token-by-token generation.

External memory. For passages d_i from a corpus $\mathcal{D}_{\text{ext}}$, we compute and store

$$k_i^{\text{ext}} = E_k(d_i), \qquad s_i^{\text{ext}} = r_{\text{frozen}}(d_i). \tag{4}$$

We use all-MiniLM-L6-v2 for E_k and a frozen Mamba-2.7B value encoder. External keys and raw values are computed offline, with keys stored in an inverted-file approximate-nearest-neighbor index. After retrieval, learnable P_{int} and P_{ext} projections map both value types to width d_m. The key encoder, external value encoder, and index remain frozen while these projections train with the language model.

3.2 Retrieval and Associative Interaction

Two learned projections produce token-level queries, $q_t^{\text{int}} = W_q^{\text{int}} h_t$ and $q_t^{\text{ext}} = W_q^{\text{ext}} h_t$. We retrieve up to $k = 8$ items from each store. Internal retrieval uses exact dot products over $\mathcal{A}_t$; external retrieval first selects candidates through the index. Within the selected set R_t^s, where $s \in \{\text{int}, \text{ext}\}$, normalized key scores define

$$a_{t,r}^s = \text{softmax}_{r \in R_t^s}\left(\frac{(q_t^s)^\top k_r^s}{\sqrt{d_k}}\right). \tag{5}$$

Top-k selection is discrete. Gradients reach query weights and selected internal values, but not selected indices or frozen external raw values.

Internal and external retrieval use separate query projections. The internal query scores earlier input chunks, and the external query scores corpus passages. The normalized retrieval weights remain token dependent after top-k selection.

Let V_t^s contain the retrieved values projected by P_s; rows corresponding to ineligible entries are masked. Before pooling, AIM applies separate directional cross-attention updates to the internal and external value matrices:

$$\widetilde{V}_t^{\text{int}} = \text{LN}\big(V_t^{\text{int}} + \text{MHA}(V_t^{\text{int}}, V_t^{\text{ext}}, V_t^{\text{ext}})\big), \tag{6}$$

$$\widetilde{V}_t^{\text{ext}} = \text{LN}\big(V_t^{\text{ext}} + \text{MHA}(V_t^{\text{ext}}, V_t^{\text{int}}, V_t^{\text{int}})\big). \tag{7}$$

Query-conditioned pooling then yields $m_t^s = \sum_{r \in R_t^s} a_{t,r}^s \widetilde{v}_{t,r}^s$. In the internal-only model, AIM is bypassed and the retrieved internal set is pooled directly.

AIM updates individual candidates before aggregation. The two directions use separate parameters and residual paths, with padded and ineligible entries masked throughout.

The retrieval weights and the attention weights have distinct roles. The former measure query–key compatibility with the current token and determine how each retrieved set is pooled. The latter govern interactions between candidate values from the two sources. Retaining separate candidate rows through AIM allows these interactions to depend on individual passage pairs; the subsequent pooling maps each updated set to one memory vector for the residual update.

Finally, memory-specific gates control the residual update:

$$z_t^s = W_m^s m_t^s, \qquad g_t^s = \sigma\big(W_g^s [h_t; z_t^s]\big), \tag{8}$$

$$o_t = h_t + g_t^{\text{int}} \odot z_t^{\text{int}} + g_t^{\text{ext}} \odot z_t^{\text{ext}}. \tag{9}$$

We refer to the internal-only ablation as PUM-Int and the two-memory model as PUM-Full.

3.3 Implementation and Resource Accounting

The augmented variants in Table 1 share a Mamba-130M backbone and replace 20 blocks with SWA, NSA, or PUM blocks. All PUM blocks use the same frozen sentence encoder for retrieval keys. PUM-Full additionally requires a frozen 2.7B Mamba value encoder, an IVF index over corpus keys, and storage for encoded passages. These resources are excluded from the nominal backbone scale and receive no updates from the language-modeling loss. The memory projections, AIM, and gates are trainable. Table 1 therefore compares a common backbone scale without equalizing offline encoding cost, index size, or stored representations.

Both stores use 64-token chunks and return at most eight entries per query. Internal values are rebuilt for each augmented block and minibatch; external raw values are shared across blocks and projected after retrieval. Construction diagrams, pseudocode, and boundary cases are in the appendix.

3.4 Training and Cost

For each PUM block, a training forward pass independently encodes complete chunks to construct $\mathcal{M}_{\text{int}}$ and scans the full sequence to obtain H. Retrieval, AIM, and fusion are then evaluated across positions. The language-model head and next-token cross-entropy follow the complete stack of blocks. Gradients propagate through selected internal values, projections, AIM, and gates. The key encoder, external value encoder, ANN index, and discrete top-k indices remain outside the gradient path.

Chunk encoding, the sequence scan, and retrieval across positions are batched during training. During prefill, each augmented block constructs a causal store from completed prompt chunks. Decoding adds an entry when a new chunk is completed and queries the updated internal store together with the fixed external index.

For one PUM block, the SSM scan and chunk scan are $O(L)$ in sequence length. Exact internal scoring compares L queries with $M \approx L/L_c$ keys, requiring $O(L^2 d_k / L_c)$ work; it can be tiled to avoid materializing the entire score matrix. AIM costs $O(Lkd_m^2 + Lk^2 d_m)$ including its dense attention projections; external lookup depends on the ANN candidate budget. During decoding, exact internal search grows with the number of completed chunks. PUM-Net is therefore not strictly linear end to end, even though its recurrent computation remains linear.

4 Experiments

The experiments evaluate length extrapolation, exact in-context recall, long-context question answering, and block-level efficiency. Tables report point estimates for the evaluated configurations.

4.1 Long-Range Language Modeling

Protocol. We fine-tune the Mamba-130M variants separately on PG-19 (Rae et al., 2020), ProofPile (Azerbayev et al., 2022), and CodeParrot (von Werra, 2021), using 100M training tokens per dataset. Training uses 4k-token windows, AdamW at 10^{-4}, global batch size 128, weight decay 0.1, and gradient clipping at 1.0. Evaluation scores the final 100 tokens of ten maximally spaced windows per example. Table 1 compares models at the same nominal backbone scale: Mamba, Mamba+SWA, Mamba+NSA, PUM-Int, and PUM-Full through 32k.

External memory uses English Wikipedia for PG-19, arXiv mathematics and Wikipedia mathematics for ProofPile, and Python files from The Stack outside the CodeParrot training data (Kocetkov et al., 2023). The frozen 2.7B value encoder, index, and stored representations are additional resources used by PUM-Full. Appendix D describes corpus selection.

Scoring the final 100 tokens of each window concentrates evaluation on predictions with an available long prefix. Sampling ten spaced windows per example covers multiple positions within a document. The reported perplexities therefore summarize these tail positions within the sampled windows.

Table 1: Perplexity after fine-tuning with 4k-token windows (lower is better). Evaluation scores the final 100 tokens of ten windows per example. All variants use the nominal Mamba-130M backbone scale; PUM-FULL additionally uses corpus evidence and a frozen 2.7B encoder.

Dataset	Length	Mamba	+SWA	+NSA	PUM-INT	PUM-FULL
PG-19	4k	16.69	16.45	16.38	16.12	**15.96**
	8k	17.35	17.01	16.92	16.77	**16.63**
	16k	17.79	17.55	17.43	16.98	**16.86**
	32k	26.54	25.82	25.60	22.32	**22.23**
ProofPile	4k	4.52	4.38	4.31	4.01	**3.53**
	8k	4.60	4.53	4.49	4.17	**3.61**
	16k	5.37	5.21	5.14	4.84	**4.40**
	32k	28.02	27.10	26.83	27.54	**26.50**
CodeParrot	4k	4.61	4.50	4.48	4.47	**1.11**
	8k	5.34	5.22	5.19	5.17	**3.18**
	16k	7.31	7.20	7.15	7.18	**4.25**
	32k	26.01	25.62	25.50	25.14	**22.50**

Results. PUM-FULL achieves the lowest PPL in all twelve dataset–length settings in Table 1. Internal retrieval improves on Mamba in every setting, although it underperforms NSA on ProofPile at 32k. The incremental effect of external memory varies across domains. At 32k, PPL decreases from 22.32 to 22.23 on PG-19, from 27.54 to 26.50 on ProofPile, and from 25.14 to 22.50 on CodeParrot. The PG-19 difference at this length is small. Figure 2 reports relative reductions, with absolute values given in the table. Appendix J reports the 2.8B comparison and the AIM ablation against simple gated fusion.

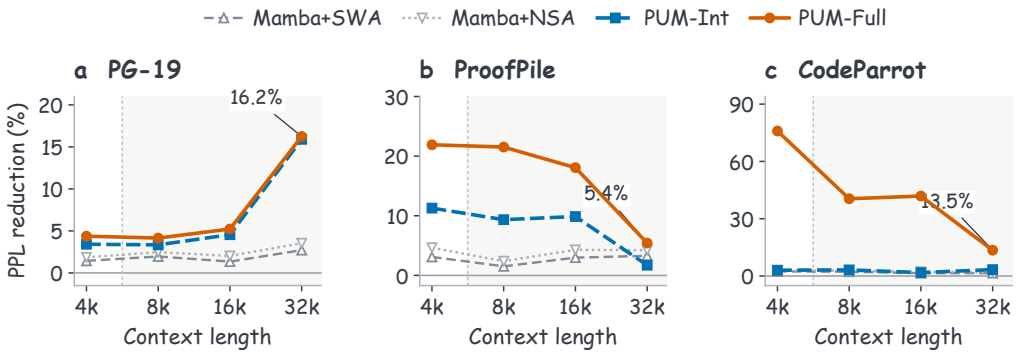


Figure 2: Relative PPL reduction from Mamba, computed as $100(1 - \mathrm{PPL}_{\mathrm{model}}/\mathrm{PPL}_{\mathrm{Mamba}})$ (higher is better). Panels use different vertical scales. Dashed lines separate the 4k training window from longer contexts; endpoint labels show PUM-FULL at 32k. Table 1 and Figure 7 give absolute values.

Figure 3 shows the training-loss trajectories, including the separation among model traces on ProofPile and CodeParrot. The horizontal axis denotes training step; the plotted intervals are specified in Appendix D.

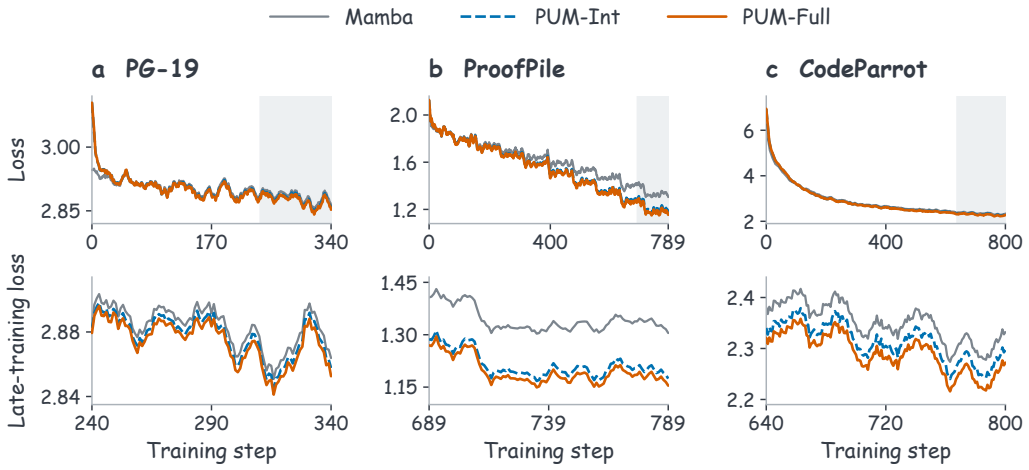


Figure 3: Training loss versus training step on PG-19, ProofPile, and CodeParrot. Upper panels show the plotted training intervals; lower panels enlarge the shaded late-training intervals. Comparisons are within each dataset. Appendix G describes the figure digitization.

4.2 Exact In-Context Recall

Following DeciMamba (Ben-Kish et al., 2025), we insert a five-digit passkey at five relative depths in WikiText distractor text (Merity et al., 2017). Models are trained for one epoch with 4k contexts and evaluated at lengths from 1k to 64k. External memory is disabled. Each of the 35 length–depth combinations contains one binary trial.

At 130M, PUM-Int succeeds in 33/35 trials, compared with 25/35 for Mamba, 30/35 for DeciMamba, and 32/35 for MambaExtend (Figure 4). For contexts longer than the 4k training window, the corresponding counts are 18/20 for PUM-Int, 19/20 for MambaExtend, and 10/20 for Mamba. Thus the aggregate advantage over MambaExtend does not extend to this subset. At 1.4B, PUM-Int succeeds in 30/35 trials overall and 17/20 beyond 4k. Table 6 provides counts by context-length group.

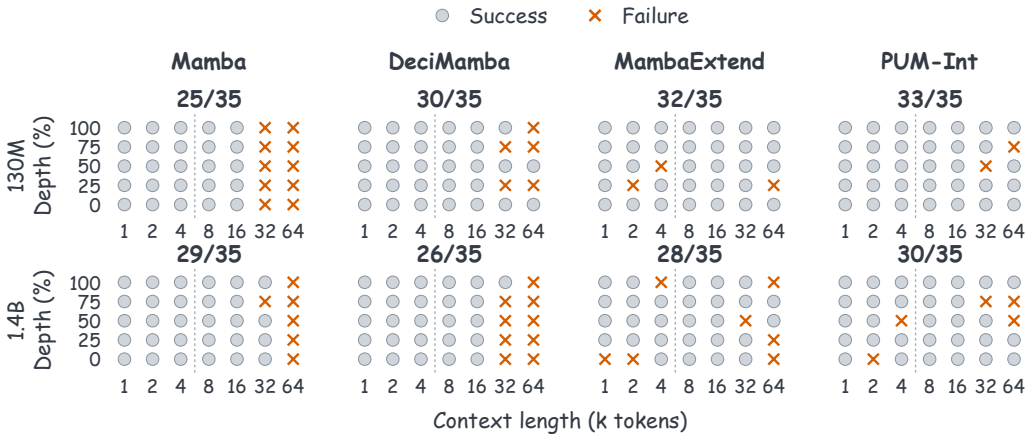


Figure 4: Passkey retrieval outcomes. Circles indicate success and crosses failure; each grid contains 35 single-trial settings. Vertical separators mark the boundary between contexts at or below 4k and longer contexts. At 130M, success beyond 4k is 18/20 for PUM-Int and 19/20 for MambaExtend (Table 6).

4.3 LONG-CONTEXT QUESTION ANSWERING

We evaluate task-specific fine-tuning of Mamba-2.8B on seven LongBench tasks (Bai et al., 2024). Among the configurations in Table 2, only PUM-FULL receives the top retrieved English Wikipedia page. Table 2 reports LongBench-E scores by length bin and separate LongBench (LB) scores.

Table 2: Task-specific QA scores on LongBench-E (length bins) and LongBench (LB). Mean LB is the macro-average over seven tasks. Bin boundaries use English word counts; task types and mean lengths are listed in Appendix F. Dashes denote tasks absent from LongBench-E. Only PUM-FULL receives a retrieved Wikipedia page in this table.

Task	Mamba-2.8B				PUM-INT				PUM-FULL			
	0–4k	4–8k	8k+	LB	0–4k	4–8k	8k+	LB	0–4k	4–8k	8k+	LB
2WikiMultihopQA	8.47	2.34	1.18	4.53	10.72	5.48	2.93	9.58	**12.42**	**6.84**	**3.63**	**11.46**
HotpotQA	5.77	2.02	0.53	2.28	7.08	3.36	2.03	4.24	**8.88**	**4.53**	**2.71**	**5.69**
MuSiQue	–	–	–	1.23	–	–	–	2.38	–	–	–	**3.47**
NarrativeQA	–	–	–	1.27	–	–	–	2.13	–	–	–	**3.08**
Qasper	7.52	5.17	2.14	6.09	9.14	8.29	2.73	9.28	**11.18**	**10.07**	**3.63**	**11.84**
MultiFieldQA-en	19.28	6.73	2.93	12.46	24.12	12.28	4.64	18.87	**27.76**	**14.47**	**6.24**	**21.73**
TriviaQA	11.46	6.38	4.52	5.83	14.23	10.07	7.17	10.36	**16.87**	**12.82**	**9.58**	**13.69**
Mean LB				4.81				8.12				**10.14**

On all seven LB tasks, PUM-FULL scores higher than PUM-INT, which in turn scores higher than Mamba. The macro-average increases from 4.81 to 8.12 and 10.14, respectively. The same ordering holds in the available LongBench-E bins; MuSiQue and NarrativeQA are absent from that evaluation. The observed 2.02-point difference compares the corpus-enabled configuration with internal retrieval, combining additional evidence access with its representation and integration.

Controlling retrieved evidence. With the same retrieved Wikipedia page, top-eight retrieval, Mamba-2.8B backbone, and 4k-context fine-tuning, PUM-FULL exceeds direct state addition in all nine LongBench-E bins and all three LB benchmarks (Table 7). LB F1 rises from 5.18 to 11.46 on 2WikiMultihopQA, 2.95 to 5.69 on HotpotQA, and 13.92 to 21.73 on MultiFieldQA-en. The control holds evidence access fixed while comparing complete integration configurations.

4.4 BLOCK EFFICIENCY

We measure forward time, backward time, and peak allocated memory for a single PUM-Net, NSA, or Flash-Attention block on an H100 at lengths of 16k–64k. Offline memory construction is excluded. A separate 64k comparison with Mamba (Table 10) reports forward times of 6.02 and 10.83 ms, and backward times of 37.77 and 69.11 ms, for Mamba and PUM-Net, respectively. These block times include internal retrieval and AIM fusion. The augmented block incurs additional computation relative to the recurrent backbone.

PUM-Net uses less block time and memory than the two attention blocks (Figure 5). The comparison measures computation within the augmented block. Offline construction of the approximately 100M-token store takes about 6.5 minutes for keys on one H100, 10 minutes for values on eight H100s, and less than one minute for indexing on eight H100s (Table 8).

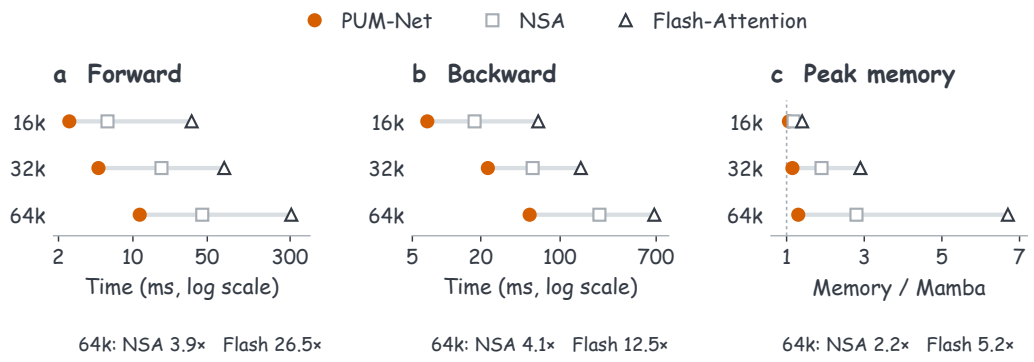


Figure 5: Single-block computation on one H100. Each row denotes a context length. Time is shown on logarithmic axes and memory relative to Mamba. Labels below the panels give NSA/PUM and Flash/PUM ratios at 64k. Plotted coordinates are approximate digitizations of the original vector figures. The measurements cover block computation.

4.5 Discussion and Limitations

PUM-Net treats evidence retention and recurrent state evolution as complementary components of sequence modeling. Keeping completed context chunks available for causal retrieval reduces reliance on the current state to preserve every detail, while a reusable offline corpus memory supplies representations across sequences. AIM couples the retrieved candidate sets before aggregation, enabling cross-source conditioning within the model's representation space. The main limitation is computational: exact internal retrieval scales with the number of stored chunks, increasing its cost as the observed context grows.

5 Conclusion

Long-context modeling exposes a mismatch between compact recurrent computation and the need to recover evidence whose relevance becomes clear only later. PUM-Net addresses this mismatch by separating state evolution from representations retained for subsequent retrieval. Its causal input memory provides an explicit route back to completed context chunks, while its offline corpus memory makes related representations reusable across sequences. This organization expands the evidence available to a prediction beyond what survives in the current recurrent state.

Separate retrieval paths accommodate the growing input memory and the reusable corpus store. Their interaction occurs before pooling, so each candidate can incorporate information from the other source before the gates regulate the residual update. Independent chunk encoding and one full-sequence scan preserve batched SSM computation, while memory construction, retrieval, and fusion account for the additional work.

After fine-tuning with 4k-token windows, internal retrieval lowers perplexity relative to Mamba in all twelve evaluated settings through 32k, and the corpus-enabled configuration achieves the lowest perplexity in these comparisons. In the 130M passkey evaluation beyond 4k, internal retrieval recovers the key in 18 of 20 single-trial settings, compared with 10 for Mamba. With the retrieved page held fixed, learned memory integration also outperforms direct state addition on three QA benchmarks. Together, causal context access and corpus reuse offer an effective complement to the compact recurrent state.

AI USE STATEMENT

We used AI tools to assist with language polishing, literature discovery, and manuscript preparation. These tools helped refine wording, organize the presentation, identify potentially relevant references, and prepare LaTeX and figures. We also used AI assistance to obtain feedback on the methodology, refine mathematical notation and pseudocode, and examine interpretations of the reported results. This assistance did not include running the model experiments or generating experimental measurements. We take full responsibility for the final content of the paper, including its claims, results, and citations.

REPRODUCIBILITY STATEMENT

Section 3 describes the memory construction, causal retrieval, and fusion mechanism. Algorithms 1 and 2 specify the batched forward computation and autoregressive inference procedure. Appendix A defines the notation and causal boundary cases, and Appendix C gives the computational costs by execution phase. Appendix D documents the language-modeling datasets, training budget, optimization settings, and tail-window evaluation protocol. Appendices E and F describe the passkey and question-answering evaluations. Appendix G specifies block-level timing, offline memory construction costs, and the figure digitization procedure. Appendix J provides the additional scale comparisons and fusion ablation, while Appendix I summarizes the evaluation scope and the interpretation of the configuration comparisons.

REFERENCES

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob L. Menick, Sebastian Borgeaud, Andy Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikołaj Bińkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karén Simonyan. Flamingo: a visual language model for few-shot learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 23716–23736. Curran Associates, Inc., 2022. doi: 10.52202/068431-1723. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/960a172bc7fbf0177ccccbb411a7d800-Paper-Conference.pdf.

Zhangir Azerbayev, Edward Ayers, and Bartosz Piotrowski. proof-pile. Hugging Face Hub dataset, 2022. URL https://huggingface.co/datasets/hoskinson-center/proof-pile. Accessed July 29, 2026.

Seyedarmin Azizi, Souvik Kundu, Mohammad Erfan Sadeghi, and Massoud Pedram. MambaExtend: A training-free approach to improve long context extension of Mamba. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=LgzRo1RpLS.

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 Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL https://arxiv.org/abs/2004.05150.

Assaf Ben-Kish, Itamar Zimerman, Shady Abu-Hussein, Nadav Cohen, Amir Globerson, Lior Wolf, and Raja Giryes. DeciMamba: Exploring the length extrapolation potential of Mamba. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=iWSl5Zyjjw.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego De Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2206–2240. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/borgeaud22a.html.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 11079–11091. Curran Associates, Inc., 2022. doi: 10.52202/068431-0805. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/47e288629a6996a17ce50b90a056a0e1-Paper-Conference.pdf.

Yingfa Chen, Xinrong Zhang, Shengding Hu, Xu Han, Zhiyuan Liu, and Maosong Sun. Stuffed Mamba: Oversized states lead to the inability to forget. In *Conference on Language Modeling (COLM)*, 2025. URL https://openreview.net/forum?id=CdRauNXD1w.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1285. URL https://aclanthology.org/P19-1285/.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. 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. 10041–10071. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/dao24a.html.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2023. URL https://arxiv.org/abs/2312.00752.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *The Tenth International Conference on Learning Representations, ICLR 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3929–3938. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/guu20a.html.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=HklBjCEKvH.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Yacine Jernite, Margaret Mitchell, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. The Stack: 3 TB of permissively licensed source code. *Transactions on Machine Learning Research*, 2023, 2023. URL https://openreview.net/forum?id=pxpbTdUEpD.

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 H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

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, 2024. URL https://arxiv.org/abs/2403.19887.

Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/9dcb88e0137649590b755372b040afad-Paper.pdf.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SylKikSYDH.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, December 2022. doi: 10.1145/3530811. URL https://doi.org/10.1145/3530811.

THUDM. Longbench: Benchmark documentation and task statistics. Official benchmark repository, 2023. URL https://github.com/THUDM/LongBench/tree/main/LongBench.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30, pp. 5998–6008. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Leandro von Werra. Training CodeParrot from scratch. Hugging Face Blog, December 2021. URL https://huggingface.co/blog/codeparrot. Published December 8, 2021; accessed July 29, 2026.

Qinwen Wang, Jieping Luo, Aoxiang Qin, Ruoyu Zhao, Jianxiong Tang, Wei Zhang, Zhichao Lu, and Luziwei Leng. Mamba with hierarchical memory: Solving representation bottleneck in long sequence modeling. *arXiv preprint arXiv:2608.02347*, 2026. URL https://arxiv.org/abs/2608.02347.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 74530–74543. Curran Associates, Inc., 2023. doi: 10.52202/075280-3259. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/ebd82705f44793b6f9ade5a669d0f0bf-Paper-Conference.pdf.

Yuhuai Wu, Markus Norman Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *The Tenth International Conference on Learning Representations, ICLR 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=TrjbxzRcnf-.

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. HELMET: How to evaluate long-context models effectively and thoroughly. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=293V3bJbmE.

Danlong Yuan, Jiahao Liu, Bei Li, Huishuai Zhang, Jingang Wang, Xunliang Cai, and Dongyan Zhao. Remamba: Equip mamba with effective long-sequence modeling. *arXiv preprint arXiv:2408.15496*, 2024. URL https://arxiv.org/abs/2408.15496.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1126. URL https://aclanthology.org/2025.acl-long.1126/.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: Transformers for longer sequences. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 17283–17297. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/c8512d142a2d849725f31a9a7a361ab9-Paper.pdf.

A Notation and Memory Construction

Table 3: Notation used in the paper and appendix.

Symbol	Meaning
B, L, d	batch size, sequence length, block width
L_c, M	chunk length and number of chunks
d_k, d_m	key width and common memory width
k	maximum retrieved entries per memory
X_{I_j}, U_{I_j}	token text and block-input representations of chunk j
h_t, o_t	preliminary and memory-augmented states
$\mathcal{M}_{\text{int}}$	sequence-specific causal memory
$\mathcal{M}_{\text{ext}}$	external key-value memory
R_t^s	entries retrieved from memory s at t

All experiments use $L_c = 64$ and retrieve at most $k = 8$ entries from each enabled memory. The frozen all-MiniLM-L6-v2 encoder forms keys from token text X_{I_j}. Internal values are computed from block-input representations U_{I_j} using the parameters of the corresponding block. The readout r_θ denotes the terminal chunk representation selected for storage.

For a hidden state that predicts the next token after position t, only chunks whose final token is at or before t are eligible. Current incomplete and future chunks are masked before top-k. The internal retrieved set has size $\min(k, |\mathcal{A}_t|)$; padded rows are masked, and an empty set produces an update hard-set to zero after projection and gating. If either set is empty, AIM is bypassed for that position and the available set is pooled without cross-attention.

External keys and values are constructed offline from 64-token corpus passages. A frozen Mamba-2.7B encoder produces the values, which are indexed by passage identifier. Keys are stored in an inverted-file index. After retrieval, trainable projections map values from each source to the common AIM width. The key encoder, external value encoder, and corpus values remain frozen during language-model training.

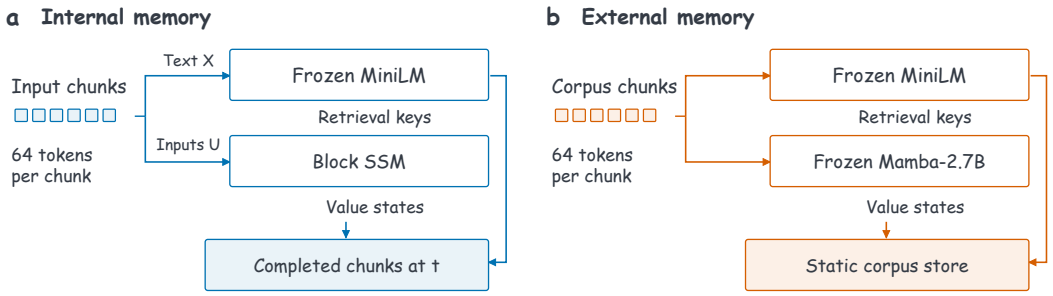


Figure 6: Construction of the two memories. Internal entries (left) are computed from current-input chunks during the forward pass. External entries (right) are encoded from corpus passages offline. Both use sentence-encoder keys and terminal chunk representations as values.

A.1 Block-Level Data Flow

Figure 1 depicts the PUM block in the main text. The equations in Section 3 specify AIM on the retrieved top-k sets before query-weighted pooling and gated fusion.

A.2 Causal Availability and Boundary Cases

A complete chunk j spans $(j-1)L_c + 1, \ldots, jL_c$. The output at position t predicts token $t+1$, and eligibility is $jL_c \leq t$. A final suffix shorter than L_c stays in the input buffer; the end of a finite

Algorithm 1 Batched forward computation of one PUM block

Require: Token text X, block input U, chunk length L_c, external index $\mathcal{I}_{\text{ext}}$
Form complete chunk intervals $I_1, \ldots, I_M$; keep shorter suffixes outside memory
Form keys $E_k(X_{I_j})$ and batched values $r_\theta(U_{I_j})$
Run one full-sequence Mamba scan to obtain $H = [h_1, \ldots, h_L]$
for $t = 1$ to L in parallel **do**
    Form internal and external queries from h_t
    Mask internal entries with $jL_c > t$
    Retrieve up to k internal entries and k external entries
    Project raw values and mask ineligible rows
    Apply bidirectional AIM if both sets are nonempty; otherwise bypass it
    Pool each updated set using its query-key weights
    Fuse the memory signals with h_t using learned gates
end for
Return $O = [o_1, \ldots, o_L]$ to the following network layer

prompt does not itself complete that chunk. This fixed-boundary convention makes the training mask and streaming insertion rule agree and excludes padding from memory.

For $t < L_c$, the internal update is zero. If fewer than k chunks are available, retrieval uses all eligible entries and masks unused rows before interaction and pooling. A newly completed chunk is available when predicting the token after its boundary. If either store yields no entries, AIM is bypassed for that position and the nonempty set is pooled directly.

External entries are static and do not use the sequence mask. Their availability is controlled by the corpus and ANN index associated with an example. In the LongBench setting, the index contains passages from the single Wikipedia page selected for that question; in language modeling, it contains the domain-specific corpus described in Appendix D.

B Algorithms

Algorithm 1 states the causal training computation. Independent chunk encoding and the full-sequence scan are batched, after which all positions perform retrieval and fusion.

The language-model head and next-token loss are applied after the full stack of blocks. Top-k indices, the key encoder, the external value encoder, and the ANN index are outside the gradient path. Gradients pass through selected internal values, value projections, AIM, pooling weights, and gates. For PUM-Int, external retrieval and AIM are bypassed, and query-weighted internal pooling feeds the internal gate directly.

At inference, completed prompt chunks initialize the internal memory. An incomplete prompt suffix remains outside the store until it reaches L_c tokens. Generated tokens advance the recurrent state one step at a time; whenever a new chunk completes, the model appends its key and block-specific value to internal memory. The insertion rule and the training mask use the same next-token convention.

C Complexity by Execution Phase

Let L be the sequence length, L_c the chunk length, $M = \lfloor L/L_c \rfloor$ the number of completed chunks, k the retrieval width, d_k the query width, and d_m the memory width. The following costs apply to one augmented block.

Training. The full-sequence scan and independent chunk encoding are linear in L. Exact internal retrieval scores L queries against at most M keys, requiring $O(LMd_k) = O(L^2 d_k / L_c)$ arithmetic. Tiling reduces the memory required for the score matrix without changing this operation count. AIM requires $O(Lkd_m^2 + Lk^2 d_m)$ computation, accounting for dense attention projections and pairwise interactions. External ANN lookup depends on the number of probed lists and candidates. Transfer volume is proportional to the number and raw width of the retrieved values.

Algorithm 2 Autoregressive inference: layer-aware data flow

Require: Prompt token sequence X, external store
Prefill each network layer in order
At each PUM layer, encode complete block-input chunks and apply causal retrieval/fusion
Retain recurrent caches, token suffix, and each PUM layer's input-state suffix buffer
while the stopping criterion is false **do**
  Emit the next token from the final layer's language-model head
  Append the emitted token to the token suffix buffer
  for each network layer in order **do**
    Update this layer's recurrent cache and preliminary output
    At a PUM layer, append the layer input to its input-state suffix buffer
    if a full chunk has completed at a PUM layer **then**
      Append its text key and block-specific value readout to that layer's memory
      Consume the completed chunk from this layer's input-state buffer
    end if
    At a PUM layer, retrieve and fuse eligible entries for the next-token prediction
  end for
  Discard completed token chunks; retain the incomplete token suffix
end while

Table 4: Resource construction and gradient flow.

Component	Built or updated	Gradient path
Sentence key encoder	Frozen, shared	No
Internal values	Per block and batch	Yes, if selected
External value encoder	Offline, frozen	No
IVF external index	Offline	No
Value projections	During training	Yes
AIM and gates	During training	Yes
Top-k indices	Each query	No
Similarity weights	Each query	Yes

Prompt prefill. Each augmented block encodes completed prompt chunks in a batch and performs one scan of its full input sequence. Exact internal retrieval has the same worst-case cost, $O(L^2 d_k / L_c)$, as in training. Corpus construction occurs offline; ANN lookup and value transfer remain part of prefill.

Autoregressive decoding. For fixed model width, the recurrent update has constant cost per generated token. At generation step r, internal retrieval compares the query with M_r completed chunks at cost $O(M_r d_k)$, and AIM costs $O(k d_m^2 + k^2 d_m)$. A new chunk value is encoded whenever the accumulated token buffer completes a chunk. The recurrent state has constant size, but exact retrieval cost increases as the internal memory grows.

D Language Modeling

Data and optimization. PG-19 uses its official splits (Rae et al., 2020). The ProofPile experiment samples 10,000 training, 1,000 validation, and 1,000 test examples from the corresponding splits (Azerbayev et al., 2022). CodeParrot uses sampled sets of 100,000, 1,000, and 2,000 files because the dataset has no fixed split (von Werra, 2021). Training examples are sampled randomly, without manual selection by difficulty. Each model is fine-tuned on 100M tokens per dataset with 4k-token windows, a fixed learning rate of 10^{-4}, global batch size 128 through accumulation, AdamW weight decay 0.1, and gradient clipping at 1.0. Evaluation takes ten maximally spaced windows per example and scores the final 100 tokens of each window.

Mamba+SWA, Mamba+NSA, and PUM-Net share the Mamba-130M backbone, with 20 blocks replaced by the corresponding augmented block. The 130M designation refers to the backbone.

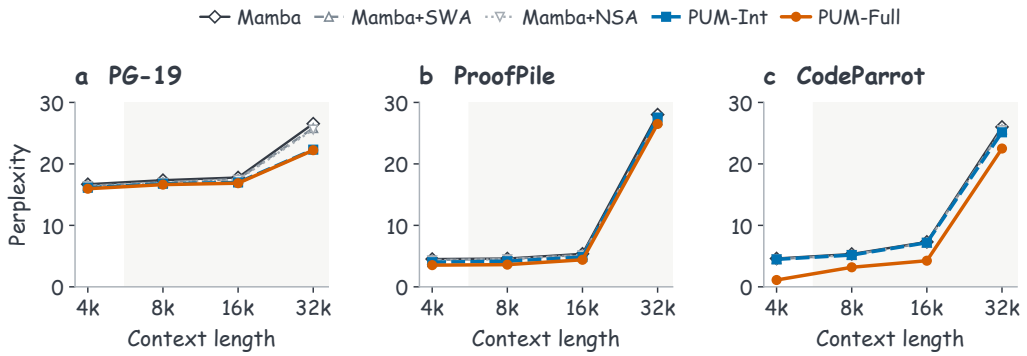


Figure 7: Absolute perplexity on a shared linear scale, complementing the relative comparisons in Figure 2. Values are identical to Table 1. All methods are evaluated after 4k-window fine-tuning.

Table 5: Language-modeling domains and external-memory sources.

Task	Training/evaluation text	External source
PG-19	Books	English Wikipedia
ProofPile	Mathematical text	arXiv math + Wiki
CodeParrot	Python repositories	The Stack (Python)

Among the configurations in Table 1, only PUM-FULL uses the frozen Mamba-2.7B value encoder and external store. The tables report configuration-level point estimates.

The language-modeling evaluation covers contexts from 4k to 32k.

External memory uses English Wikipedia for PG-19, arXiv mathematics papers and Wikipedia mathematics pages for ProofPile, and Python files from The Stack for CodeParrot (Kocetkov et al., 2023). For PG-19, the source separation is between pre-1919 Project Gutenberg books and a Wikipedia snapshot. For ProofPile, the external subset supplies mathematical reference material, including definitions, theorems, and Wikipedia mathematics pages. For CodeParrot, the Stack subset targets files outside the CodeParrot training data. The experimental store contains approximately 100M tokens from selected source material, rather than the complete raw corpus dumps. These choices specify the corpus provenance and file-selection protocol. The internal causal mask separately enforces prefix-only retrieval during both training and evaluation.

For each evaluation length, ten windows are placed at the largest constant stride supported by the example. Perplexity is computed from the final 100 tokens of each window, reducing the contribution of positions close to a fresh window boundary. Figure 2 reports the results over 4k–32k.

Training-loss trajectories. Figure 3 plots training loss against training step. The displayed intervals are 0–340 for PG-19, 0–789 for ProofPile, and 0–800 for CodeParrot; the enlarged intervals are 240–340, 689–789, and 640–800, respectively. The curves compare the loss trajectories of the three configurations within each dataset. The token budget and optimization settings are specified above.

E PASSKEY RETRIEVAL

The task follows DeciMamba (Ben-Kish et al., 2025). A five-digit code is inserted into WikiText distractor text (Merity et al., 2017) at relative depths of 0%, 25%, 50%, 75%, and 100%. Context lengths are 1k, 2k, 4k, 8k, 16k, 32k, and 64k. All models are fine-tuned for one epoch with 4k contexts, and a generation succeeds only when it reproduces the code verbatim.

Each of the 35 length–depth combinations contains one binary trial. At 130M, success counts are 25 for Mamba, 30 for DeciMamba, 32 for MambaExtend, and 33 for PUM-INT; at 1.4B, they are 29,

Table 6: Successful passkey trials by context length. The first two groups partition the 35 settings; 32–64k is a subset of the > 4k group. Each setting contains one binary trial.

Scale	Model	≤ 4k	> 4k	32–64k	All
130M	Mamba	15/15	10/20	0/10	25/35
130M	DeciMamba	15/15	15/20	5/10	30/35
130M	MambaExtend	13/15	19/20	9/10	32/35
130M	PUM-INT	15/15	18/20	8/10	33/35
1.4B	Mamba	15/15	14/20	4/10	29/35
1.4B	DeciMamba	15/15	11/20	1/10	26/35
1.4B	MambaExtend	12/15	16/20	6/10	28/35
1.4B	PUM-INT	13/15	17/20	7/10	30/35

26, 28, and 30, respectively. Each setting uses a distinct prompt, and the reported counts summarize success over this fixed evaluation grid.

At 130M, PUM-INT has two fewer failures than MambaExtend at 2k/4k and one additional failure beyond 4k. Their aggregate ranking therefore reverses when evaluation is restricted to extrapolation lengths. Relative to Mamba, success beyond 4k increases from 10/20 to 18/20 at 130M and from 14/20 to 17/20 at 1.4B. The comparisons report differences in successful settings across the fixed length–depth grid.

F LONGBENCH

The task categories and mean input lengths are: 2WikiMultihopQA (multi-document, 4,887 words), HotpotQA (multi-document, 9,151), MuSiQue (multi-document, 11,214), NarrativeQA (single-document, 18,409), Qasper (single-document, 3,619), MultiFieldQA-en (single-document, 4,559), and TriviaQA (few-shot, 8,209).

The three variants in Table 2 initialize from the same instruction-tuned Mamba-2.8B checkpoint and apply task-specific fine-tuning (Bai et al., 2024). Evaluation uses LongBench and LongBench-E (THUDM, 2023). For PUM-FULL, one English Wikipedia page is retrieved per question, divided into 64-token passages, and encoded into a question-specific store. Mamba and PUM-INT in Table 2 receive no additional page.

The 0–4k, 4–8k, and 8k+ columns are LongBench-E length bins; LB denotes evaluation on the separate LongBench collection. MuSiQue and NarrativeQA are absent from LongBench-E. Consequently, LB is not a weighted average of the adjacent bins. Mean LB is the macro-average over seven tasks. Bin boundaries and mean input lengths use the benchmark's English word-count convention (THUDM, 2023); the model's training window is measured in tokens.

The difference between PUM-FULL and PUM-INT includes both access to the retrieved page and the architecture used to encode and integrate it. In table order, the differences are 1.88, 1.45, 1.09, 0.95, 2.56, 2.86, and 3.33 F1 points. These are comparisons between complete configurations with different evidence access.

F.1 Comparison with the Same Retrieved Evidence

The naive hidden-state retrieval control uses the same top-ranked Wikipedia page as PUM-Full. Both use a Mamba-2.8B backbone and 4k-context fine-tuning. The control retrieves the top eight encoded states and adds them directly to the Mamba hidden state without learned fusion. Thus the page and retrieval width are held fixed, while the integration mechanisms differ.

Table 7: QA F1 with matched external evidence. Naive state addition and PUM-Full receive the same retrieved Wikipedia page and use top-eight state retrieval. All models are fine-tuned with 4k contexts. Length bins belong to LongBench-E; LB is the separate LongBench evaluation.

Task	Length / set	Mamba	Naive state addition	PUM-Full
2WikiMultihopQA	0–4k	8.47	8.12	**12.42**
	4–8k	2.34	4.15	**6.84**
	8k+	1.18	2.05	**3.63**
	LB	4.53	5.18	**11.46**
HotpotQA	0–4k	5.77	5.42	**8.88**
	4–8k	2.02	3.10	**4.53**
	8k+	0.53	1.45	**2.71**
	LB	2.28	2.95	**5.69**
MultiFieldQA-en	0–4k	19.28	18.85	**27.76**
	4–8k	6.73	9.80	**14.47**
	8k+	2.93	4.10	**6.24**
	LB	12.46	13.92	**21.73**

Direct addition lowers F1 relative to Mamba in all three 0–4k bins but improves it in the six longer-context bins. PUM-Full exceeds direct addition in every reported condition, including gains of 6.28, 2.74, and 7.81 F1 points on the three LB evaluations. With the retrieved page and retrieval width held fixed, learned integration outperforms direct state addition. Table 12 separately compares AIM with a variant that retains learned gating while omitting cross-attention.

G Efficiency Benchmark

Figure data and precision. Perplexity curves use the numerical table entries, and passkey grids show the binary outcomes of the evaluated settings. The training-loss trajectories and efficiency coordinates are digitized from the original vector plots, preserving their paths and bar geometry without smoothing. Digitized efficiency coordinates are approximate; Table 10 gives the directly measured block times for Mamba and PUM-Net.

The efficiency figure compares one representative block for PUM-Net, Native Sparse Attention (Yuan et al., 2025), and Flash-Attention (Dao et al., 2022) on one NVIDIA H100. Sequence lengths are 16k, 32k, and 64k; we measure forward wall time, backward wall time, and peak allocated accelerator memory. The PUM block times include internal retrieval and AIM fusion. Offline external encoding and index construction are reported separately below.

At 64k, NSA and Flash-Attention require 3.9 and 26.5 times the PUM-Net forward time, 4.1 and 12.5 times its backward time, and 2.2 and 5.2 times its peak memory, respectively. These are block-level ratios. A complete serving pipeline additionally combines external ANN lookup, host–device transfer, prefill, and autoregressive decoding; its aggregate cost is outside the reported benchmark scope.

Offline external-memory construction. We construct the external store from a targeted corpus of approximately 100M tokens. Keys are encoded by all-MiniLM-L6-v2, values by frozen Mamba-2.7B, and the resulting keys are indexed for approximate nearest-neighbor retrieval. Table 8 reports the component wall times and hardware allocations. The store is built once and reused during model training; these figures describe construction, not online retrieval or autoregressive serving.

Table 8: Offline construction of an approximately 100M-token external store. Wall times are approximate component measurements with the listed hardware allocations; the rows are not summed into a sequential pipeline latency.

Component	Hardware	Wall time
Key encoding (MiniLM)	1 H100	$\sim$6.5 min
Value encoding (Mamba-2.7B)	8 H100s	$\sim$10 min
ANN index construction	8 H100s	$<$1 min

H Encoded-State Retrieval and Text Retrieval

Appending retrieved passages increases the target model's token workload. Encoder-based cross-attention, as in RETRO, processes the evidence separately but still requires contextual evidence representations. PUM-Net computes corpus values offline and retrieves them inside SSM blocks. This avoids repeated corpus-value encoding while requiring storage, indexing, transfer, and the use of a frozen representation.

PUM-Net constructs token-level queries at each augmented block. Its retrieval schedule therefore differs from interfaces that query once per question or once per chunk. Offline corpus encoding and runtime lookup contribute separate costs, in addition to the recurrent computation. Table 2 compares configurations with different evidence access. Table 7 instead holds the retrieved page and retrieval width fixed and compares direct state addition with learned integration. Both use encoded states; the latter comparison does not evaluate a text-concatenation interface.

I Experimental Scope

The tables report point estimates for each configuration. The passkey grid contains one binary outcome per length–depth setting; it summarizes the displayed test cases rather than a distribution over repeated samples. Source separation specifies how the external corpora were selected, while document-level overlap rates are not quantified by these evaluations.

The causal mask excludes incomplete and future chunks from internal retrieval during both training and evaluation. AIM operates on the retrieved candidate sets before pooling. Algorithms 1 and 2 describe the training and autoregressive inference procedures. Language-modeling comparisons cover 4k–32k contexts, while the passkey and block-timing evaluations extend to 64k.

The internal-only and corpus-enabled comparisons evaluate complete memory configurations. The same-page QA control in Table 7 holds external evidence and retrieval width fixed, and the PG-19 state-addition control in Table 11 uses the same backbone and external store. These controls evaluate learned integration relative to direct addition. Table 12 separately examines cross-memory interaction by comparing AIM with simple gated fusion on PG-19 at 4k–32k. Timing evidence comprises measured block costs, including internal retrieval and AIM, and separately reported offline construction costs.

J Additional Scale and Ablation Checks

The 2.8B PG-19 comparison fine-tunes Pythia, Mamba, and PUM-Int on the same 100M-token sample using 4k-context windows. The following tables report this scale comparison, direct Mamba block timing, a matched-store state-addition control, and an AIM ablation against simple gated fusion.

Table 9: Additional 2.8B PG-19 tail-window perplexity with a 100M-token fine-tuning budget. Dashes denote PPL above 100; these entries are excluded from numerical comparisons.

PG-19 model (2.8B)	4k	8k	16k	32k
Pythia	9.85	–	–	–
Mamba	9.17	15.60	–	–
PUM-INT	9.12	14.35	14.80	18.55

Table 10: Single-block timing on an H100 at the 130M scale, including internal retrieval and AIM for PUM-Net. PUM-Net requires approximately 1.80× the Mamba forward time and 1.83× its backward time. Offline construction and full-model serving are outside this measurement.

64k H100 single-block time	Mamba	PUM-Net
Forward (ms)	6.02	10.83
Backward (ms)	37.77	69.11

Table 11: Fusion configurations on PG-19 at 4k. Naive addition and PUM-FULL use the same Mamba-130M backbone and external memory. Naive retrieval directly adds projected top-k external states without joint fusion training.

Setting	Condition	Reported PPL
PG-19, 4k	Mamba	16.69
	Naive untrained external-state addition	17.85
	PUM-FULL	15.96

Interpretation. At 4k, untrained state addition increases PPL from 16.69 to 17.85, whereas PUM-FULL yields 15.96 with the same external store. The following ablation examines the contribution of cross-memory interaction with learned gating retained.

J.1 AIM VERSUS SIMPLE GATED FUSION

Comparison. We compare AIM with simple gated fusion on PG-19 using the Mamba-130M backbone. The simple-gating variant retains both internal and external memory sources and the final learned gating layer, while removing the bidirectional cross-attention between the retrieved candidate sets. AIM introduces cross-memory interaction between the retrieved candidate sets before pooling and gated integration. Table 12 reports perplexity at 4k, 8k, 16k, and 32k context lengths.

Table 12: AIM versus simple gated fusion on PG-19 with the Mamba-130M backbone. Both memory sources and the final learned gate are retained in the simple-gating variant; cross-attention is removed. Lower PPL is better. Relative reduction is $100(1 - \mathrm{PPL}_{\mathrm{AIM}}/\mathrm{PPL}_{\mathrm{gate}})$.

Context	Simple gating	AIM	PPL reduction	Relative reduction
4k	16.63	**15.96**	0.67	4.03%
8k	18.33	**16.63**	1.70	9.27%
16k	19.57	**16.86**	2.71	13.85%
32k	25.16	**22.23**	2.93	11.65%

Results. AIM yields lower perplexity at every evaluated length, with relative reductions of 4.03%–13.85%. The absolute improvement increases from 0.67 PPL at 4k to 2.93 PPL at 32k. These comparisons support a contribution from cross-memory interaction beyond the retained gating mechanism across the evaluated context lengths.

The comparison complements the same-page QA control in Table 7. That control evaluates how the complete learned integration configuration uses the same retrieved page, whereas Table 12 compares two fusion variants with both memory sources active. Together, the results support the use of learned integration and provide a separate ablation of the cross-attention component on PG-19.