LSTMEM: HIERARCHICAL LONG SHORT-TERM ONLINE MEMORY FOR LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models increasingly serve as long-horizon assistants and agents, where they must both accumulate information across interactions and make the relevant parts available when later requests depend on them. Existing compact online memories typically use a single persistent state both to accumulate history and to serve readout, so what the memory stores cannot be controlled separately from what it exposes to the current computation. We propose LSTMem, an LSTM-inspired online memory that instead equips each layer of a frozen LLM with two matrix-valued states: a cell state that accumulates history and a hidden state whose readouts correct the backbone's attention. Input and forget gates control what the cell stores, while an output gate separately controls what the cell exposes through the hidden state. LSTMem further connects memory across depth through forward hidden-state propagation and block-end feedback, and uses higher-layer reconstruction gradients to refine lower-layer cell states before rebuilding hidden states from shallow to deep layers. Across memory benchmarks on Qwen3-4B-Instruct, LSTMem consistently improves MemoryAgentBench, LoCoMo, and HotpotQA over the plain backbone. Comparisons further show that the LSTM-based memory formulation outperforms an associative-memory counterpart, while removing cross-layer hidden-memory propagation degrades performance. These results demonstrate the benefits of separating memory accumulation from memory expression and organizing memory hierarchically across model depth.

1 INTRODUCTION

Multi-session dialogue and multi-step agent tasks require large language models (LLMs) to remember what earlier interactions established and to recall it when a later request depends on it (Hu et al., 2026; Wu et al., 2024; Yang et al., 2024a). Keeping the full history in context becomes increasingly costly as interactions accumulate, and longer inputs can also degrade performance even when the relevant evidence is retrieved correctly (Du et al., 2025b; Laban et al., 2026). Compact online memory offers another way to retain history in a fixed-size state that persists across interactions, with its effectiveness depending on how new information is accumulated and made available to the model during computation (Zhoubian et al., 2026).

Recurrent matrix memories realize such a state by continually updating a compact matrix as tokens are processed (Sun et al., 2023; Dao & Gu, 2024). Gated Linear Attention and Gated DeltaNet regulate these updates with learned gates (Yang et al., 2023; 2025b), and δ-Mem brings a gated delta-rule state to a frozen backbone, turning its readout into attention corrections (Lei et al., 2026). In each case, however, the matrix that accumulates history is also the one being read, and although gates can modulate each write or readout, the state exposed to the model is the stored history itself. Retention and exposure are therefore coupled, so content kept for later stays exposed even when it is irrelevant to the current step, and useful information, such as memory already held by lower layers, can be exposed only by storing it again in the limited state. A separate hidden state would let one matrix accumulate history while another forms the representation the model reads. LSTM makes this distinction explicit through a cell state that accumulates information and a hidden state that exposes a gated transformation of that content (Hochreiter & Schmidhuber, 1997; Gers et al., 2000; Beck et al., 2024). The cell is updated additively under input and forget gates, while the output gate controls its contribution to the hidden state, separating retention and writing from memory expression. Motivated by this design, we propose LSTMem, which equips a frozen Transformer backbone with

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.

a matrix-valued cell memory C and hidden memory D at each memory layer (Figure 1). During history processing, the current input first queries the available hidden memory to generate low-rank corrections Δq and Δo to the attention query and output, before gated writes update the states for subsequent use.

With the widespread adoption of deep Transformer architectures, we extend this separation across model depth so that deeper memories can build on representations formed in shallower layers. The hidden memory $D_t^{(\ell-1)}$ is passed to layer ℓ, where it participates in both the construction of new memory and the readout used to guide attention. This forward path is complemented by feedback at history-block boundaries: reconstruction gradients from higher layers are used to correct lower-layer cell states, and the hidden states are then rebuilt from shallow to deep layers. Because the correction is derived from key–value associations in the history, it requires neither answer labels nor changes to model weights.

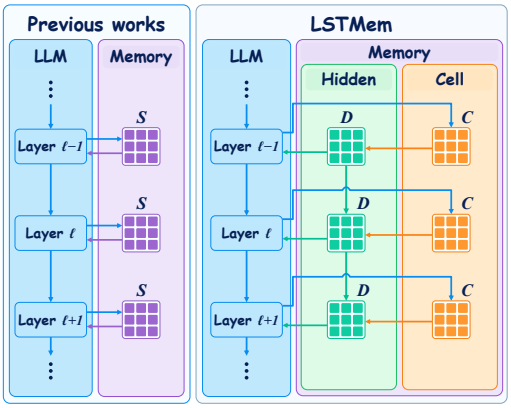

Figure 1: **Comparison of memory designs.** LSTMem separates accumulation in the cell state C from memory expression in the hidden state D, and passes $D_t^{(\ell-1)}$ to layer ℓ.

We evaluate LSTMem on MemoryAgentBench (Hu et al., 2026) and HotpotQA (Yang et al., 2018) using Qwen3-4B-Instruct, Qwen3-8B (Yang et al., 2025a), and SmolLM3-3B (Bakouch et al., 2025), and further test conversational memory and general capabilities on LoCoMo (Maharana et al., 2024), IFEval (Zhou et al., 2023), and GPQA-Diamond (Rein et al., 2023) with Qwen3-4B-Instruct. Relative to the published backbone results, LSTMem improves MemoryAgentBench and HotpotQA F1 on all three models (Table 1). On Qwen3-4B-Instruct, it gains 15.54 points on MemoryAgentBench, 12.81 on LoCoMo, and 12.12 on HotpotQA F1; its IFEval score is only 0.19 points lower and its GPQA-Diamond score 3.03 points higher (Tables 1–3). Removing hidden-memory propagation across layers lowers MemoryAgentBench by 1.92 and 1.77 points on two training seeds, with the largest drops on test-time learning (6.21 and 8.00 points; Table 4). These gains come at a modest cost, as LSTMem retains on average 92.10% of the decoding speed of δ-Mem and takes only 5.9% to 6.5% longer end-to-end with 2,048-token answers (Appendix D).

Our contributions are summarized as follows:

- We propose LSTMem, an online memory for frozen pretrained language models that uses separately gated states C, D to accumulate history and supply low-rank attention corrections through hidden-memory readouts.
- We introduce a layer-hierarchical memory structure in which $D_t^{(\ell-1)}$ connects successive memory layers, together with block-end feedback that uses higher-layer reconstruction gradients to revise lower-layer cells and then rebuilds the hidden states.
- Extensive experiments on MemoryAgentBench, LoCoMo, and HotpotQA across three LLMs demonstrate the effectiveness of LSTMem. Comparisons further indicate that organizing memory hierarchically across depth is beneficial.

2 Preliminaries

Associative memory. At token position t, each memory head of width r maintains a fixed-size state $S_t \in \mathbb{R}^{r\times r}$ that stores associations between keys and values $k_t, v_t \in \mathbb{R}^r$ (Yang et al., 2024b; Lei et al., 2026). Holding k_t, v_t fixed, a gradient descent step on $\frac{1}{2}\|Sk_t - v_t\|_2^2$ with respect to S at S_{t-1} gives the delta-rule update

$$S_t = S_{t-1} + \eta_t\big(v_t - S_{t-1}k_t\big)k_t^\top,$$

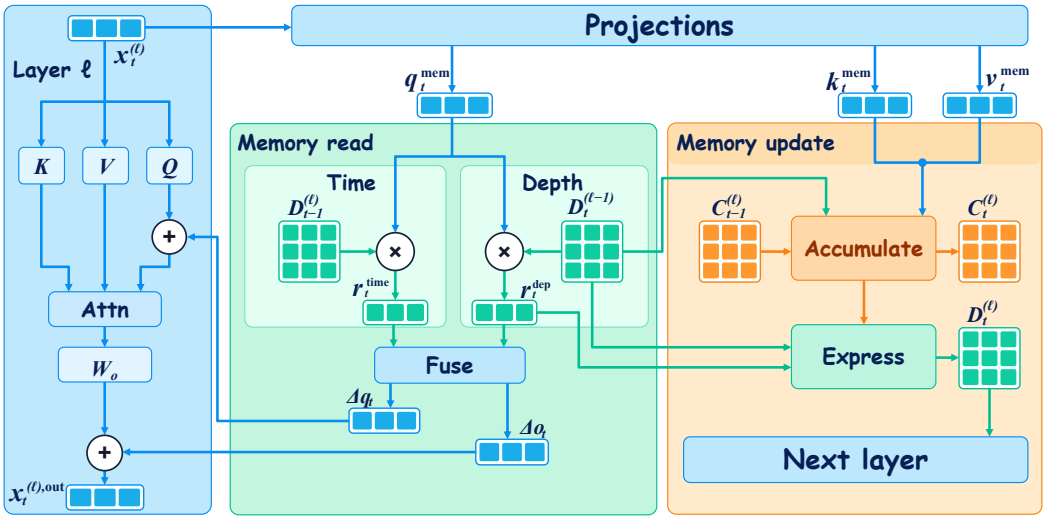

Figure 2: **Memory readout and update in LSTMem.** At each layer, LSTMem reads memory from earlier tokens and the layer below to guide the frozen model's attention. Gated updates accumulate history in C and control the memory expressed through D, which is carried across tokens and layers.

where $\eta_t \geq 0$ is the scalar write rate. The residual directs writes toward associations that the current state predicts incorrectly.

LSTM accumulation and expression. LSTM separates accumulated content in a cell state c_t from its expression through a hidden state s_t (Hochreiter & Schmidhuber, 1997; Gers et al., 2000; Beck et al., 2024):

$$c_t = f_t \odot c_{t-1} + i_t \odot \widetilde{c}_t, \qquad s_t = o_t \odot \tanh(c_t). \tag{1}$$

Here $\widetilde{c}_t$ is the candidate content and $\odot$ denotes elementwise multiplication. The input and forget gates i_t, f_t regulate writing and retention, while the output gate o_t controls memory expression. At layer ℓ, LSTMem adopts this separation with matrix states $C_t^{(\ell)}$ for accumulation and $D_t^{(\ell)}$ for readout and propagation across layers (Section 3).

3 LSTMem

Figure 2 shows how LSTMem reads memory to guide attention (Section 3.1) and updates memory through gated accumulation and expression (Section 3.2). We then describe cross-layer feedback for refining memory at block boundaries (Section 3.3; Figure 3). Section 3.4 presents the training and inference procedure. Further details are provided in Appendix C.

3.1 Memory readout and attention correction

Given the input $x_t^{(\ell)} \in \mathbb{R}^d$ to the attention module at position t in layer ℓ, LSTMem projects it to low-dimensional memory vectors $q_t^{\text{mem}}, k_t^{\text{mem}}, v_t^{\text{mem}} \in \mathbb{R}^{d_s}$:

$$q_t^{\text{mem}} = \mathrm{n}\left(W_q^{\text{mem}} x_t^{(\ell)}\right), \quad k_t^{\text{mem}} = \mathrm{n}\left(W_k^{\text{mem}} x_t^{(\ell)}\right), \quad v_t^{\text{mem}} = W_v^{\text{mem}} x_t^{(\ell)}. \tag{2}$$

The vector q_t^{mem} reads memory, while k_t^{mem} and v_t^{mem} participate in the update in Section 3.2. Memory has n_h heads of vector width r, so $d_s = n_h r$; $\mathrm{n}(\cdot)$ applies $\tanh$ followed by ℓ_2 normalization within each head. Memory projection parameters are specific to each layer. We omit layer superscripts on projections and intermediate vectors when describing a fixed memory layer.

Each layer maintains $C^{(\ell)}, D^{(\ell)} \in \mathbb{R}^{n_h \times r \times r}$ for historical accumulation and memory expression. Before writing the current position, the same q_t^{mem} reads two memories: $D_{t-1}^{(\ell)}$ carried from the

preceding position in the current layer, and $D_t^{(\ell-1)}$ formed at the current position in the preceding layer. These correspond to the temporal and depth read paths in Figure 2. Let $j \in \{1, \ldots, n_h\}$ index memory heads, with $[j]$ selecting the corresponding vector or matrix and concat_j concatenating head outputs:

$$r_t^{\mathrm{time}} = \mathrm{concat}_j\left(D_{t-1}^{(\ell),[j]} q_t^{\mathrm{mem},[j]}\right), \quad r_t^{\mathrm{dep}} = \mathrm{concat}_j\left(D_t^{(\ell-1),[j]} q_t^{\mathrm{mem},[j]}\right), \\ u_t = W_{\mathrm{read}}\left[r_t^{\mathrm{time}}; r_t^{\mathrm{dep}}\right] + b_{\mathrm{read}}. \tag{3}$$

Fusing the two readout vectors lets the current layer use its own history together with memory newly formed at the same position in the preceding layer. Using one-based layer indices, we set $D_t^{(0)} = \mathbf{0}$ as the depth input to the first memory layer. Two projections map the fused signal $u_t \in \mathbb{R}^{d_s}$ to attention corrections:

$$\Delta q_t = \frac{\alpha}{r} W_{\Delta q} u_t, \qquad \Delta o_t = \frac{\alpha}{r} W_{\Delta o} u_t. \tag{4}$$

The projection matrices $W_{\Delta q}$ and $W_{\Delta o}$ are learned, and α is a fixed scale constant. We add Δq_t to the model's original query before native query normalization and positional encoding (Su et al., 2024), allowing memory to influence access to the explicit context; keys and values follow the original computation. After the output projection, we add Δo_t to the attention result, letting memory directly correct the output representation.

3.2 Gated accumulation and expression

LSTMem uses C to accumulate historical content and D to provide the representation for reading and propagation across layers. This separation allows content updates and memory expression to be controlled separately.

In the memory update illustrated in Figure 2, lower-layer memory provides a reference for the current layer's write. We read $D_t^{(\ell-1)}$ with k_t^{mem} to obtain p_t^{dep} and construct the write residual e_t from its prediction:

$$p_t^{\mathrm{dep}} = \mathrm{concat}_j\left(D_t^{(\ell-1),[j]} k_t^{\mathrm{mem},[j]}\right), \quad \hat{v}_t = \widetilde{W}_{\mathrm{pred}} p_t^{\mathrm{dep}} + b_{\mathrm{pred}}, \quad e_t = v_t^{\mathrm{mem}} - \hat{v}_t. \tag{5}$$

The prediction readout p_t^{dep} comes from k_t^{mem}. The gates use r_t^{dep}, read with q_t^{mem} in Section 3.1, together with the current input representation z_t and write residual e_t. The same layer input $x_t^{(\ell)}$ also produces $z_t \in \mathbb{R}^{d_s}$:

$$z_t = W_x x_t^{(\ell)} + b_x, \quad [i_t; f_t; o_t; g_t] = \sigma\left(\widetilde{W}_{\mathrm{gate}}[z_t; r_t^{\mathrm{dep}}; e_t] + b_{\mathrm{gate}}\right). \tag{6}$$

Here σ is the elementwise sigmoid function, and $i_t, f_t, o_t, g_t \in (0,1)^{d_s}$ are the input, forget, output, and depth gates. The input and forget gates control writing and retention in C; the output and depth gates control how local accumulation and lower-layer memory are expressed in D. For each head, the two state updates are

$$C_t^{(\ell),[j]} = \mathrm{diag}\left(f_t^{[j]}\right) C_{t-1}^{(\ell),[j]} + \mathrm{diag}\left(i_t^{[j]}\right) e_t^{[j]} \left(k_t^{\mathrm{mem},[j]}\right)^{\top}, \tag{7}$$

$$D_t^{(\ell),[j]} = \mathrm{diag}\left(o_t^{[j]}\right) \tanh\left(C_t^{(\ell),[j]}\right) + \mathrm{diag}\left(g_t^{[j]}\right) D_t^{(\ell-1),[j]}. \tag{8}$$

Lower-layer memory thus participates both in constructing the write candidate and in forming the expressed state, establishing a continuous memory pathway across depth. Updated C and D persist across positions, while D also passes to the next layer at the same position. Effective projection parameterizations and parallel computation within a block are detailed in Appendix C.

3.3 Cross-layer feedback

We process history in consecutive segments, referred to as history blocks. Within a block, memory flows from shallow to deep layers, so a lower layer cannot yet use higher-layer reconstruction errors

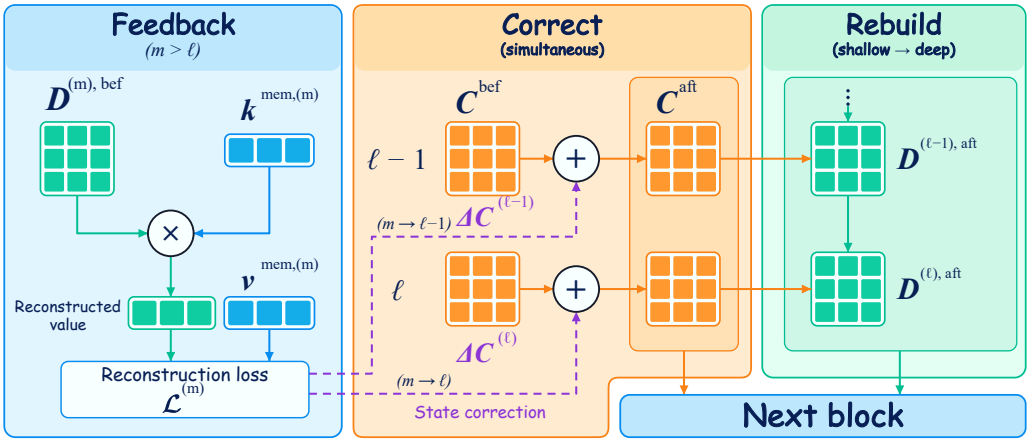

Figure 3: **Cross-layer feedback.** After each history block, lower-layer cells are corrected simultaneously using the reconstruction errors of all higher layers (one source layer $m > \ell$ is shown), and the expressed states are then rebuilt from shallow to deep with cached gates for the next block. Sampled keys, values, gates, and other cells are held fixed in the correction gradient.

for the same history when it writes. LSTMem therefore introduces feedback at block boundaries to correct lower-layer cells using higher-layer errors (Figure 3).

Let b index the current history block, and use superscripts bef and aft for terminal states before and after feedback. All layers share a set of sampled positions $\mathcal{I}_b$ from this block. Using the memory keys and values already computed at these positions, each layer reconstructs its associations from its terminal expressed state:

$$\mathcal{L}_b^{(\ell)} = \frac{1}{2|\mathcal{I}_b| n_h r} \sum_{t \in \mathcal{I}_b} \sum_{j=1}^{n_h} \left\| D_b^{(\ell),\mathrm{bef},[j]} k_t^{\mathrm{mem},(\ell),[j]} - v_t^{\mathrm{mem},(\ell),[j]} \right\|_2^2. \tag{9}$$

Associations from early positions are tested after later writes have modified the state, so the loss measures whether they remain recoverable at the block boundary. Sharing $\mathcal{I}_b$ across layers ties all reconstruction errors to the same historical tokens, while each layer is tested on its own projected keys and values, so every loss reflects recall in that layer's own key–value space.

These historical associations provide feedback without additional answer labels. For layer ℓ, the correction uses reconstruction errors only from higher layers $m > \ell$. The inner derivative treats terminal cell states as independent variables and follows only the depth-state recurrence in Eq. 8, with sampled keys, values, terminal gates, and the other terminal cell states held fixed. With L memory layers, the correction is summarized by

$$G_b^{(\ell)} = \frac{\partial}{\partial C_b^{(\ell),\mathrm{bef}}} \sum_{m=\ell+1}^{L} \mathcal{L}_b^{(m)}, \quad C_b^{(\ell),\mathrm{aft}} = C_b^{(\ell),\mathrm{bef}} + \Delta C_b^{(\ell)}. \tag{10}$$

The correction $\Delta C_b^{(\ell)}$ follows the negative direction of $G_b^{(\ell)}$, with headwise normalization, a fixed step size, and a cell RMS scale bounded below by one. All corrections are evaluated at the same pre-feedback states and committed simultaneously after numerical checks. The highest layer receives no error from above, so its cell remains unchanged.

Correcting C also requires updating the expressed state D used for reading and depth propagation. Starting from the pre-feedback states and cached gates, we rebuild D from shallow to deep layers along the existing depth connection. Each rebuilt state incorporates both its local cell correction and the change in the preceding layer's expressed state. The highest-layer D can therefore change even though its cell is unchanged. The corrected C and D initialize memory for the next history block. Feedback changes only the memory carried forward; model parameters and outputs already computed within the block remain unchanged. The gradient recurrence, headwise correction rule, and incremental rebuild are given in Appendix C.4.

3.4 TRAINING AND INFERENCE

LSTMem trains the adapter with answer cross-entropy while keeping the base model frozen. Each training example starts from zero memory: we process its history, update C and D, and apply block-end feedback before predicting the final assistant turn using the resulting states. During the training answer pass, memory writing and feedback are disabled, and the historical KV cache is not carried into answer computation. Both read paths query the cached terminal hidden states of the corresponding memory layers, while C and D remain fixed across answer tokens. The stored memory influences attention through the readout path in Section 3.1. The reconstruction objective in Section 3.3 corrects these states and is not added as an auxiliary term to the answer loss.

Training begins on QASPER (Dasigi et al., 2021) with the history computation graph retained. Answer-loss gradients can therefore pass through memory readout into historical writing and feedback computation, supervising how the memory is constructed as well as how it is used. Holding keys, values, and gates fixed in Section 3.3 applies only to the inner feedback derivative; it does not stop answer-loss gradients through these computations. We then continue training on the Long dataset, which contains 71 synthetic histories ranging from 44,074 to 98,340 tokens (Appendix C.5). Histories in the Long dataset are still written and corrected in blocks, but history processing runs without gradient tracking, and states are detached between blocks. For these examples, the answer loss trains the adapter through memory readout and attention correction.

At inference, all model parameters are fixed. Historical writing and block-end feedback continue to form C and D, whose readouts influence subsequent answers. State updates during answer generation and the history retained in the prompt follow the corresponding evaluation protocols (Lei et al., 2026); memory is reset between independent conversations (Appendix C.5). The training objective, training data, initialization, and evaluation read/write settings are given in Appendices B and C.5.

4 EXPERIMENTS

We evaluate LSTMem on MemoryAgentBench (MAB; Hu et al., 2026) and HotpotQA (Yang et al., 2018) using Qwen3-4B-Instruct, Qwen3-8B (Yang et al., 2025a), and SmolLM3-3B (Bakouch et al., 2025). On Qwen3-4B-Instruct, we also evaluate LoCoMo (Maharana et al., 2024), IFEval (Zhou et al., 2023), and GPQA-Diamond (Rein et al., 2023). The main 4B model backbone is frozen, and only the memory adapter is trained. The baseline results in Tables 1–3 are taken from Lei et al. (2026), with MAB averages recomputed from the published family scores using question-count weights. LoCoMo averages are also weighted by question count. Table 6 additionally reports δ-Mem results after our second-stage training on the Long dataset. We first present the main results on the memory benchmarks and on general capabilities in Section 4.1 and then ablate the depth coupling, the choice of memory layers, the training stages, and block-end feedback in Section 4.2. On Qwen3-4B-Instruct, LSTMem retains 92.10% of δ-Mem's equivalent single-step decoding speed on average. Its end-to-end times are 5.9%–6.5% longer than δ-Mem's for 8K-, 32K-, and 128K-token histories with 2,048-token answers as detailed in Appendix D. More detailed training and evaluation settings are provided in Appendix B.

Baselines. We compare LSTMem with four families of memory methods on the same backbones, namely external memory with BM25 RAG (Lewis et al., 2020; Robertson & Zaragoza, 2009), LLMLingua-2 (Pan et al., 2024; Jiang et al., 2023), and MemoryBank (Zhong et al., 2024); parametric memory with Context2LoRA (Hu et al., 2021; Back et al., 2026) and MemGen (Zhang et al., 2026); auxiliary memory with MLP Memory (Wei et al., 2026); and online memory with δ-Mem (Lei et al., 2026). For δ-Mem, we report its multi-state write (MSW) variant, which obtains the best MAB and LoCoMo averages among its variants on Qwen3-4B-Instruct and is also the most direct comparison, as it likewise keeps parallel $r \times r$ states per layer, corrects the attention query and output with the same r and α, and is trained on QASPER with the same settings as our first stage. Unlike LSTMem, MSW reads the same states it writes and has no depth coupling or block-end feedback.

Implementation details. On Qwen3-4B-Instruct, LSTMem attaches memory to all 36 layers with $n_h = 4$ heads of width $r = 8$ and $\alpha = 16$. The persistent C/D state for one sequence contains 18,432 values, occupying 36 KiB in BF16 independently of history length. Block-end feedback sums the reconstruction losses of all higher layers over at most 32 evenly spaced positions per history block and runs in both training and inference. Both training stages use AdamW with batch size 32,

Table 1: **Main results across three base models.** Bold marks the highest score within each model group; subscripts give point changes from its unadapted model. Baseline scores are from Lei et al. (2026). HotpotQA is scored by exact match (EM) and token-level F1 against the gold answer. AR, TTL, LRU and SF denote accurate retrieval, test-time learning, long-range understanding and selective forgetting. AR and SF are scored by accuracy, TTL by classification accuracy and movie-recommendation Recall@5, and LRU by summarization F1 and Detective QA accuracy. MAB averages use 2,000/700/171/800 question weights for AR/TTL/LRU/SF; baseline averages are approximated from rounded family scores.

Model	HotpotQA ↑		MemoryAgentBench ↑				
	EM	F1	**Avg.**	AR	TTL	LRU	SF
Qwen3-4B-Instruct	42.35	56.00	29.54	35.30	26.14	**47.08**	14.37
External Memory							
+ BM25 RAG	$40.35_{-2.00}$	$52.83_{-3.17}$	$24.43_{-5.11}$	$32.20_{-3.10}$	$9.74_{-16.40}$	$37.86_{-9.22}$	$15.00_{+0.63}$
+ LLMLingua-2	$36.93_{-5.42}$	$50.03_{-5.97}$	$15.63_{-13.91}$	$21.45_{-13.85}$	$1.43_{-24.71}$	$38.45_{-8.63}$	$8.62_{-5.75}$
+ MemoryBank	—	—	$17.65_{-11.89}$	$22.65_{-12.65}$	$7.67_{-18.47}$	$36.36_{-10.72}$	$9.88_{-4.49}$
Parametric Memory							
+ Context2LoRA	$37.85_{-4.50}$	$50.88_{-5.12}$	$32.53_{+2.99}$	$40.00_{+4.70}$	$29.86_{+3.72}$	$25.15_{-21.93}$	$17.75_{+3.38}$
+ MemGen	$5.36_{-36.99}$	$16.27_{-39.73}$	$29.61_{+0.07}$	$34.85_{-0.45}$	$28.45_{+2.31}$	$44.30_{-2.78}$	$14.38_{+0.01}$
Auxiliary Memory							
+ MLP Memory	$10.94_{-31.41}$	$25.83_{-30.17}$	$28.80_{-0.74}$	$35.35_{+0.05}$	$26.00_{-0.14}$	$31.19_{-15.89}$	$14.38_{+0.01}$
Online Memory							
+ δ-Mem (MSW)	$46.86_{+4.51}$	$60.47_{+4.47}$	$38.85_{+9.31}$	$44.40_{+9.10}$	$47.29_{+21.15}$	$41.55_{-5.53}$	$17.00_{+2.63}$
+ **LSTMem (Ours)**	$\mathbf{53.76}_{+11.41}$	$\mathbf{68.12}_{+12.12}$	$\mathbf{45.08}_{+15.54}$	$\mathbf{55.70}_{+20.40}$	$\mathbf{47.50}_{+21.36}$	$37.59_{-9.49}$	$\mathbf{18.00}_{+3.63}$
Qwen3-8B	32.48	41.42	31.87	45.10	12.79	**48.22**	12.00
Parametric Memory							
+ Context2LoRA	$36.22_{+3.74}$	$49.19_{+7.77}$	$30.53_{-1.34}$	$43.15_{-1.95}$	$10.05_{-2.74}$	$34.04_{-14.18}$	$16.13_{+4.13}$
Online Memory							
+ δ-Mem (MSW)	$40.15_{+7.67}$	$51.34_{+9.92}$	$32.66_{+0.79}$	$45.55_{+0.45}$	$14.95_{+2.16}$	$44.60_{-3.62}$	$13.38_{+1.38}$
+ **LSTMem (Ours)**	$\mathbf{53.76}_{+21.28}$	$\mathbf{67.49}_{+26.07}$	$\mathbf{34.61}_{+2.74}$	$\mathbf{46.80}_{+1.70}$	$\mathbf{18.74}_{+5.95}$	$41.70_{-6.52}$	$\mathbf{16.50}_{+4.50}$
SmolLM3-3B	1.67	14.40	11.08	12.57	5.53	30.72	8.00
Parametric Memory							
+ Context2LoRA	$30.28_{+28.61}$	$44.39_{+29.99}$	$14.23_{+3.15}$	$16.08_{+3.51}$	$2.86_{-2.67}$	$36.77_{+6.05}$	$14.75_{+6.75}$
Online Memory							
+ δ-Mem (MSW)	$31.61_{+29.94}$	$46.77_{+32.37}$	$16.81_{+5.73}$	$18.10_{+5.53}$	$8.32_{+2.79}$	$\mathbf{39.63}_{+8.91}$	$\mathbf{16.12}_{+8.12}$
+ **LSTMem (Ours)**	$\mathbf{46.55}_{+44.88}$	$\mathbf{59.65}_{+45.25}$	$\mathbf{19.89}_{+8.81}$	$\mathbf{22.10}_{+9.53}$	$\mathbf{14.33}_{+8.80}$	$37.98_{+7.26}$	$15.38_{+7.38}$

BF16, and a cosine schedule with 10% warm-up. The first stage runs 70 updates on QASPER at a learning rate of 2×10^{-4}, with histories capped at 8,192 tokens. The second stage runs 26 updates at 5×10^{-5} with a fresh optimizer and writes each history in 2,048-token blocks, bringing the full schedule to only 96 updates. Further implementation and training details are given in Appendix C.

4.1 Main results

With Qwen3-4B-Instruct as the base model, LSTMem achieves the highest average scores on MAB and LoCoMo among the methods compared in Tables 1 and 2: 45.08 on MAB and 53.60 F1 on LoCoMo, 6.23 and 4.48 points above the published δ-Mem MSW results. The gains concentrate on tasks that require retaining specific information from a long history and drawing on it later, the capability that motivates LSTMem. Accurate retrieval contributes 6.16 of the 6.23-point MAB gain over δ-Mem (+11.30), and on all three backbones LSTMem is higher than δ-Mem MSW in both accurate retrieval and test-time learning; on Qwen3-4B-Instruct, selective forgetting is also 1.00 point higher. The flip side is LRU, which requires a holistic reading of a long narrative rather than retaining specific details, hence, on Qwen3-4B-Instruct and Qwen3-8B, every memory-augmented method in Table 1 falls below the unadapted model on this family. LoCoMo most directly reflects

Table 2: **LoCoMo F1 (%).** Avg. is weighted over 1,540 questions. Multi, Temp., Open and Single denote multi-hop, temporal, open-domain and single-hop questions.

Model	Avg.	Multi	Temp.	Open	Single
Qwen3-4B-Instruct	40.79	38.39	32.89	10.77	48.05
+ BM25 RAG	36.68	38.12	20.34	9.99	45.47
+ LLMLingua-2	40.98	39.07	30.13	10.98	49.19
+ MemoryBank	38.14	37.88	21.76	13.35	47.31
+ Context2LoRA	48.11	37.95	34.99	16.75	60.11
+ MemGen	40.05	32.93	33.30	12.67	48.13
+ MLP Memory	26.85	32.87	16.72	8.81	30.75
+ δ-Mem (MSW)	49.12	42.57	39.31	18.12	58.59
+ LSTMem (Ours)	**53.60**	**46.25**	**40.94**	**19.92**	**64.75**

Table 3: **General-capability scores (%).** IFEval uses strict prompt-level accuracy; GPQA-D uses accuracy.

Model	IFEval ↑	GPQA-D ↑
Qwen3-4B-Instruct	81.89	39.39
+ Context2LoRA	76.71	29.29
+ MemGen	39.37	38.89
+ MLP Memory	24.95	22.73
+ δ-Mem (MSW)	81.52	37.37
+ LSTMem (Ours)	81.70	42.42

Table 4: **Depth ablation.** Δ = on $-$ off. The coupling and feedback is removed in training and inference alike.

Seed	Depth coupling	MAB Avg.	AR	TTL	LRU	SF
42	on	45.08	55.70	47.50	37.59	18.00
	off	43.16	54.30	41.29	38.21	18.00
	Δ	$+1.92$	$+1.40$	$+6.21$	-0.62	0.00
43	on	45.45	54.90	52.36	37.44	17.50
	off	43.68	54.35	44.36	37.39	17.75
	Δ	$+1.77$	$+0.55$	$+8.00$	$+0.05$	-0.25

Table 5: **Memory placement ablation on LoCoMo.** F1 (%) on Qwen3-4B-Instruct. Layer indices are one-based: even/odd select 18 alternating layers; first/middle/last select layers 1–12, 13–24 and 25–36, respectively.

Memory layout	Avg.	Multi	Temp.	Open	Single
All 36	53.60	46.25	40.94	19.92	64.75
Even 18	51.30	45.97	40.51	16.10	61.22
Odd 18	51.56	46.83	39.73	14.10	61.93
First 12	48.09	43.91	33.98	11.12	59.10
Middle 12	50.42	46.93	39.19	13.94	60.03
Last 12	50.58	47.50	41.37	14.28	59.28

the long-horizon setting behind our design, since the memory is carried across the sessions of each conversation; there, LSTMem obtains the best score among all methods in every question category, with the largest gain over δ-Mem on single-hop questions ($+6.16$). HotpotQA, which requires locating specific facts in several passages and combining them, gives the largest benchmark-level margin over δ-Mem on every backbone: LSTMem reaches 68.12 F1 on Qwen3-4B, 7.65 points above δ-Mem, and the margin widens to 16.15 and 12.88 points on Qwen3-8B and SmolLM3-3B.

LSTMem obtains the best average score on every memory benchmark. A memory interface should add history-dependent abilities while minimizing its impact on general ones (Biderman et al., 2024), so we also evaluate general capabilities on Qwen3-4B-Instruct with a fresh memory for each example (Table 3). Relative to the unadapted model, LSTMem lowers IFEval strict prompt-level accuracy by only 0.19 points and raises GPQA-Diamond accuracy by 3.03 points, the best scores among all memory methods. Because the backbone remains frozen, these changes come only from the memory interface, and detaching it recovers the original model exactly.

4.2 Ablation studies and analysis

Ablation on depth-state connection. In LSTMem, each memory layer predicts its write from the hidden memory $D_t^{(\ell-1)}$ of the layer below and carries that memory into its own expressed state (Section 3.2). Replacing $D_t^{(\ell-1)}$ with zeros in both training and inference lowers MAB by 1.92 and 1.77 points (Table 4), both larger than the 1.2-point difference between two retrainings of the coupled model with the same seed. The drop is concentrated in test-time learning (6.21 and 8.00 points); accurate retrieval also decreases on both seeds, while LRU and SF change by at most 0.62 points. Test-time learning requires learning new tasks from examples accumulated in the history rather than recalling a single fact. This matches the motivation for the hierarchy: with depth coupling, each layer writes only what the layer below does not already predict, so higher layers can build on lower-layer memory instead of duplicating it.

Table 6: **Training-stage comparison on LoCoMo.** Stage 1 trains on QASPER, and Stage 2 continues training on the Long dataset. Scores are F1 (%), with Avg. weighted by question count.

Method	Training stage	Multi-hop	Temporal	Open-domain	Single-hop	Avg.
δ-Mem	1: QASPER	42.57	39.31	18.12	58.59	**49.12**
	2: + Long	46.73	38.75	16.20	59.36	**50.06**
LSTMem	1: QASPER	43.70	39.03	18.72	60.19	**50.18**
	2: + Long	46.25	40.94	19.92	64.75	**53.60**

Table 7: **Block-end feedback ablation on LoCoMo.** F1 (%) on Qwen3-4B-Instruct. Both rows keep forward depth coupling and differ only in whether block-end feedback is applied.

Block-end feedback	Avg.	Multi	Temp.	Open	Single
On	53.60	46.25	40.94	19.92	64.75
Off	51.89	46.31	40.78	17.48	61.93

Ablation on different layers. We retrain the main configuration with memory on subsets of the 36 layers (Table 5) and take $D_t^{(\ell-1)}$ from the preceding memory-enabled layer. The full stack exceeds the best sparse layout by 2.04 points, and the average grows with the number of memory layers: 48.09–50.58 with 12 layers, 51.30–51.56 with 18 and 53.60 with 36. No sparse layout exceeds the full stack by more than 1.25 points in any individual question category, whereas every sparse layout trails it by at least 3.82 points on open-domain and 2.82 points on single-hop questions. These results support extending the memory hierarchy through the full depth of the backbone.

Ablation on training stages. Our main model continues training on the Long dataset after QASPER, whereas the published δ-Mem results use QASPER alone, so we also train δ-Mem with the same second stage (Table 6). This stage does improve δ-Mem, raising its LoCoMo average to 50.06. It helps LSTMem more: LSTMem gains 3.42 points, improves in every question category, and reaches 53.60, 3.54 points above δ-Mem after the same two stages. Even after both stages, δ-Mem remains below LSTMem after Stage 1 alone (50.18). This indicates that the advantage of LSTMem stems from the architecture, and the gap widens as training continues on long histories.

Ablation on block-end feedback. Within a history block, memory flows only from shallow to deep layers; block-end feedback adds the reverse direction, letting lower layers correct their cells with the reconstruction errors of higher layers before the next block (Section 3.3). Disabling it while keeping forward depth coupling lowers the LoCoMo average from 53.60 to 51.89 (Table 7). The loss falls on single-hop and open-domain questions (2.82 and 2.44 points), whereas multi-hop and temporal scores change by less than 0.2 points. Feedback thus accounts for 2.82 of the 6.16-point single-hop advantage of LSTMem over δ-Mem in Table 2, suggesting that it mainly sharpens the recall of individual facts stated in the conversation. Even without feedback, LSTMem remains above δ-Mem trained with the same two stages (51.89 vs. 50.06; Table 6), and feedback widens this margin to 3.54 points. The feedback correction is computed from key–value associations already in the history, without answer labels or parameter updates.

5 Conclusion

We introduced LSTMem, an online long short-term memory that complements a language model's current context with a persistent stack of gated memory cells. The cell state accumulates history, the hidden state expresses memory for current use, and hidden-state propagation conditions upper memories on lower ones; block-end feedback closes the loop by letting lower layers correct their cells with the reconstruction errors of higher layers, without answer labels or parameter updates. With the backbone frozen, LSTMem obtains the best MemoryAgentBench averages and HotpotQA scores among all compared methods on all three backbones, as well as the best LoCoMo score on Qwen3-4B-Instruct, with gains concentrated on tasks that require retaining specific information from a long history. Ablations support each design choice: removing depth coupling lowers MemoryAgentBench on both training seeds, memory on all layers gives the best LoCoMo average, disabling feedback lowers LoCoMo, and the advantage over δ-Mem persists when both are trained with the same stages. These findings motivate treating online memory as both a temporal process and a hierarchy across model depth.

AI USE STATEMENT

In this work, we used generative AI tools for [generating synthetic data sets, helping develop theoretical models or conceptual frameworks, designing research methods or experiments or providing feedback on them, implementing methods, assisting with translation]. We have not used generative AI tools for [supporting qualitative and thematic data analysis, interpreting results], and [formulating mathematical claims, providing key elements for proving mathematical claims, assisting in writing proofs, generating or refining hypotheses, cleaning and reformatting data sets] are not applicable to this work. We have reviewed all AI-assisted work. In particular, all LLM-generated code was verified and tested for correctness by two authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies online memory for an open-weight language model using public benchmark data and constructed training streams. We do not recruit human participants or collect new user interactions. A deployed persistent memory may retain information beyond a single session, so users should be able to control memory updates and reset the state.

REPRODUCIBILITY STATEMENT

Section 3 and Appendix C specify the memory updates, block-end feedback, and training recipe. Appendix B gives the evaluation protocol, and Appendix D the efficiency setup. We release our code as well.

REFERENCES

Seungju Back, Dongwoo Lee, Naun Kang, Taehee Lee, SK Hong, Youngjune Gwon, and Sungjin Ahn. Understanding LoRA as knowledge memory: An empirical analysis. *arXiv preprint arXiv:2603.01097*, 2026.

Elie Bakouch, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Lewis Tunstall, Carlos Miguel Patiño, Edward Beeching, Aymeric Roucher, Aksel Joonas Reedi, Quentin Gallouédec, et al. SmolLM3: smol, multilingual, long-context reasoner. *Hugging Face Blog*, 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. *Advances in Neural Information Processing Systems*, 37: 107547–107603, 2024.

Ali Behrouz, Zeman Li, Praneeth Kacham, Majid Daliri, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. ATLAS: Learning to optimally memorize the context at test time. *arXiv preprint arXiv:2505.23735*, 2025.

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

Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, et al. LoRA learns less and forgets less. *arXiv preprint arXiv:2405.09673*, 2024.

Guy E Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, 1990.

Tong Chen, Hao Fang, Patrick Xia, Xiaodong Liu, Ben Van Durme, Luke Zettlemoyer, Jianfeng Gao, and Hao Cheng. Generative adapter: Contextualizing language models in parameters with a single forward pass. In *International Conference on Learning Representations*, volume 2025, pp. 27450–27470, 2025.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, 2023.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pp. 35549–35562, 2024.

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

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, 2021.

Yiming Du, Wenyu Huang, Danna Zheng, Zhaowei Wang, Sebastien Montella, Mirella Lapata, Kam-Fai Wong, and Jeff Z Pan. Rethinking memory in LLM based agents: Representations, operations, and emerging topics. *arXiv preprint arXiv:2505.00675*, 2025a.

Yufeng Du, Minyang Tian, Srikanth Ronanki, Subendhu Rongali, Sravan Bodapati, Aram Galstyan, Azton Wells, Roy Schwartz, Eliu A Huerta, and Hao Peng. Context length alone hurts LLM performance despite perfect retrieval. *arXiv preprint arXiv:2510.05381*, 2025b.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945*, 2023.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

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

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? *arXiv preprint arXiv:2404.06654*, 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. *arXiv preprint arXiv:2106.09685*, 2021.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions. In *International Conference on Learning Representations*, volume 2026, pp. 156259–156291, 2026.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13358–13376, 2023.

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.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation. In *International Conference on Learning Representations*, volume 2026, pp. 54738–54778, 2026.

Jingdi Lei, Di Zhang, Junxian Li, Weida Wang, Kaixuan Fan, Xiang Liu, Qihan Liu, Xiaoteng Ma, Baian Chen, and Soujanya Poria. δ-mem: Efficient online memory for large language models. *arXiv preprint arXiv:2605.12357*, 2026.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

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

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, 2024.

Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with Infini-attention. *arXiv preprint arXiv:2404.07143*, 2024.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, et al. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 963–981, 2024.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023.

Bo Peng, Daniel Goldstein, Quentin Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Xingjian Du, Teddy Ferdinan, Haowen Hou, et al. Eagle and Finch: RWKV with matrix-valued states and dynamic recurrence. *arXiv preprint arXiv:2404.05892*, 2024.

Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. HGRN2: Gated linear RNNs with state expansion. *arXiv preprint arXiv:2404.07904*, 2024.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. GPQA: A graduate-level Google-proof Q&A benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval*, 4(1-2):1–174, 2009.

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.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

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

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

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

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

Ke Alexander Wang, Jiaxin Shi, and Emily B Fox. Test-time regression: a unifying framework for designing sequence models with associative memory. *Journal of Machine Learning Research*, 27 (169):1–41, 2026.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36:74530–74543, 2023.

Yan Wang, Dongyang Ma, and Deng Cai. With greater text comes greater necessity: Inference-time training helps long text generation. *arXiv preprint arXiv:2401.11504*, 2024a.

Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. MEMORYLLM: Towards self-updatable large language models. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 50453–50466, 2024b.

Yu Wang, Dmitry Krotov, Yuanzhe Hu, Yifan Gao, Wangchunshu Zhou, Julian McAuley, Dan Gutfreund, Rogerio Feris, and Zexue He. M+: Extending MemoryLLM with scalable long-term memory. *arXiv preprint arXiv:2502.00592*, 2025.

Rubin Wei, Jiaqi Cao, Jiarui Wang, Jushi Kai, Qipeng Guo, Bowen Zhou, and Zhouhan Lin. MLP memory: A retriever-pretrained memory for large language models. In *International Conference on Learning Representations*, volume 2026, pp. 132772–132795, 2026.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.

Zhe Xu, Jiasheng Ye, Xiaoran Liu, Xiangyang Liu, Tianxiang Sun, Zhigeng Liu, Qipeng Guo, Linlin Li, Qun Liu, Xuanjing Huang, et al. DetectiveQA: Evaluating long-context reasoning on detective novels. *arXiv preprint arXiv:2409.02465*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Bowen Yang, Bharat Venkitesh, Dwaraknath Gnaneshwar Talupuru, Hangyu Lin, David Cairuz, Phil Blunsom, and Acyr Locatelli. RoPE to NoPE and back again: A new hybrid attention strategy. *Advances in Neural Information Processing Systems*, 38:64133–64157, 2026.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024a.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *Advances in Neural Information Processing Systems*, 37:115491–115522, 2024b.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025b.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, et al. MemAgent: Reshaping long-context LLM with multi-conv RL-based memory agent. In *International Conference on Learning Representations*, volume 2026, pp. 39458–39486, 2026.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. MemGen: Weaving generative latent memory for self-evolving agents. In *International Conference on Learning Representations*, volume 2026, pp. 22555–22588, 2026.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, et al. ∞Bench: Extending long context evaluation beyond 100K tokens. *arXiv preprint arXiv:2402.13718*, 2024.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Sining Zhoubian, Dan Zhang, Evgeny Kharlamov, and Jie Tang. Memory for large language models, 2026. URL https://arxiv.org/abs/2607.25380.

APPENDIX CONTENTS

A Related Work **15**

B Evaluation protocol **16**

C Memory implementation details **16**

C.1 State configuration and attention interface 16

C.2 Effective projection blocks 16

C.3 Parallel state updates 17

C.4 Implementation of block-end feedback 17

C.5 Training and initialization 18

D Inference efficiency **20**

A RELATED WORK

Online Memory. Online memory incorporates new information by updating its contents during inference (Zhoubian et al., 2026; Sun et al., 2024; Behrouz et al., 2026; 2025; Wang et al., 2026; Packer et al., 2023). Linear attention admits a recurrent matrix-state formulation (Katharopoulos et al., 2020; Sun et al., 2023; Dao & Gu, 2024; Peng et al., 2024), connecting sequence processing to fast-weight memories that store key–value associations (Schlag et al., 2021). Recurrent matrix memories implement online updates by incorporating incoming associations into this fixed-size state (Gu & Dao, 2023; Qin et al., 2024; Liu et al., 2025; Team et al., 2025). Gated Linear Attention learns input-dependent retention gates (Yang et al., 2023), while DeltaNet corrects stored associations using value-prediction errors (Yang et al., 2024b). Gated DeltaNet combines these mechanisms to control forgetting and targeted writes (Yang et al., 2025b). δ-Mem connects associative memory to pretrained language models by converting state readouts into low-rank attention corrections (Lei et al., 2026). Although their update rules differ, these methods read from the matrix in which they accumulate historical associations. LSTMem separates these roles with a cell state C_t for historical accumulation and a hidden state D_t for reading, with input, forget, and output gates controlling how memory is written, retained, and expressed.

Long-term Memory. Long-term memory aims to keep earlier information available beyond the current context window (Zhoubian et al., 2026; Zhang et al., 2025; Du et al., 2025a). Textual memory systems such as MemoryBank retain interaction records for later retrieval (Zhong et al., 2024; Packer et al., 2023; Park et al., 2023; Chhikara et al., 2025; Xu et al., 2026). LongMem instead caches past key–value representations and retrieves them through a residual side network (Wang et al., 2023). Compressing history into a fixed-size memory provides another way to extend this horizon (Chevalier et al., 2023; Ge et al., 2023; Munkhdalai et al., 2024; Yu et al., 2026): MEMORYLLM maintains an updatable latent memory pool (Wang et al., 2024b; 2025), while xLSTM uses gated recurrent memories, including a matrix-valued cell in mLSTM (Beck et al., 2024). Other approaches encode knowledge in LoRA modules (Hu et al., 2021; Back et al., 2026; Wang et al., 2024a; Chen et al., 2025) or generate latent memory tokens during reasoning, as in MemGen (Zhang et al., 2026). Titans combines local attention with a neural memory updated at test time, using gradient-based surprise to guide memorization (Behrouz et al., 2026). LSTMem places recurrent memory alongside a pretrained Transformer and couples these memories across depth. At each token position, the hidden matrix $D_t^{(\ell-1)}$ passes to the next memory module, where it conditions both state construction and readout.

B EVALUATION PROTOCOL

MemoryAgentBench. For MemoryAgentBench (Hu et al., 2026), all 3,671 questions are scored with the per-source metrics: normalised alias accuracy for question answering, F1 for InfiniteBench summarisation (Zhang et al., 2024), and Recall@5 for movie recommendation. Family and overall scores are weighted by question count. The generation limits are 40 tokens for EventQA, 50 for LongMemEval (Wu et al., 2024) and RULER (Hsieh et al., 2024), 10 for FactConsolidation, 20 for in-context classification, 2,000 for Detective QA (Xu et al., 2024), 1,200 for summarisation and 300 for movie recommendation. The prompt requests a short answer supported by the supplied context and an explicit abstention when unsupported.

LoCoMo. For LoCoMo (Maharana et al., 2024), we use categories 1–4: multi-hop, temporal, open-domain and single-hop, totalling 1,540 questions. Category 5 is excluded as in Lei et al. (2026). Conversations are replayed session by session through the memory, which is reset before each independent conversation. Questions use the official single-prompt format and a 50-token answer limit. Scores use token-level F1 with Porter stemming and the official multi-answer handling; the total is a question-weighted mean of category scores.

HotpotQA, IFEval and GPQA-Diamond. HotpotQA (Yang et al., 2018) uses 7,405 validation questions with supporting passages, a 32-token generation limit and EM/F1 scoring. IFEval (Zhou et al., 2023) uses 541 prompts, at most 1,500 generated tokens and strict prompt-level accuracy. GPQA-Diamond (Rein et al., 2023) uses 198 questions, a JSON answer-letter prompt and at most 8,192 generated tokens. All five benchmarks use sampled decoding with temperature 0.4, top-p 0.9, top-k 10 and seed 42. General-capability evaluations initialise a fresh memory for each independent example.

Depth-ablation settings. The depth ablation trains and evaluates both settings with the frozen backbone. We report individual training seeds and do not estimate benchmark confidence intervals. The two historical retrainings mentioned in Section 4.2 provide a variability reference, not a statistical bound.

C MEMORY IMPLEMENTATION DETAILS

C.1 STATE CONFIGURATION AND ATTENTION INTERFACE

Our main Qwen3-4B-Instruct configuration attaches memory modules to all 36 Transformer layers, with $n_h = 4$ heads and $r = 8$. All backbone parameters remain frozen throughout both training stages, and only the memory adapter is trained. The denominator in the query/key normalization is bounded below by 10^{-6}. Both temporal and depth reads use direct matrix–vector products without additional read-side selection or decay. We use $\alpha = 16$, unit extra output scale and no memory-use gate. Native backbone processing is preserved, including Qwen3 key normalization (Yang et al., 2025a) and SmolLM3's layer-specific RoPE/no-RoPE choice (Bakouch et al., 2025; Su et al., 2024; Yang et al., 2026).

C.2 EFFECTIVE PROJECTION BLOCKS

The prediction and gate projections in the full parameterization receive a zero block in place of the current layer's temporal-memory input. Partition their weight matrices according to these inputs:

$$W^{\text{full}}_{\text{pred}} = [P_{\text{time}} \mid P_{\text{dep}}], \qquad W^{\text{full}}_{\text{gate}} = [G_z \mid G_{\text{time}} \mid G_{\text{dep}} \mid G_e].$$

The effective matrices used in Eqs. 5 and 6 are

$$\widetilde{W}_{\text{pred}} = P_{\text{dep}}, \qquad \widetilde{W}_{\text{gate}} = [G_z \mid G_{\text{dep}} \mid G_e].$$

They give the same forward maps:

$$W^{\text{full}}_{\text{pred}}[0; p^{\text{dep}}_t] = \widetilde{W}_{\text{pred}} p^{\text{dep}}_t,$$
$$W^{\text{full}}_{\text{gate}}[z_t; 0; r^{\text{dep}}_t; e_t] = \widetilde{W}_{\text{gate}}[z_t; r^{\text{dep}}_t; e_t].$$

Biases are unchanged. The full matrices are retained in the stored parameterization; the main-text matrices select their active blocks. Neither projection takes the current layer's temporal memory as an explicit input. In particular, initializing $W_{\text{pred}}^{\text{full}} = [I_{d_s} \mid 0_{d_s \times d_s}]$ corresponds to $\widetilde{W}_{\text{pred}} = 0$.

All memory readouts, residuals, and within-layer features $r_t^{\text{time}}, r_t^{\text{dep}}, p_t^{\text{dep}}, \hat{v}_t, e_t, u_t, z_t$ have width d_s. Let d_q denote the output width of the backbone query projection. The projection dimensions are

$$\begin{aligned} W_q^{\text{mem}}, W_k^{\text{mem}}, W_v^{\text{mem}}, W_x &\in \mathbb{R}^{d_s \times d}, \\ W_{\text{read}}, W_{\text{pred}}^{\text{full}} &\in \mathbb{R}^{d_s \times 2d_s}, \\ \widetilde{W}_{\text{pred}} &\in \mathbb{R}^{d_s \times d_s}, \\ \widetilde{W}_{\text{gate}} &\in \mathbb{R}^{4d_s \times 3d_s}, \\ W_{\text{gate}}^{\text{full}} &\in \mathbb{R}^{4d_s \times 4d_s}, \\ W_{\Delta q} &\in \mathbb{R}^{d_q \times d_s}, \\ W_{\Delta o} &\in \mathbb{R}^{d \times d_s}. \end{aligned}$$

The biases satisfy $b_x, b_{\text{read}}, b_{\text{pred}} \in \mathbb{R}^{d_s}$ and $b_{\text{gate}} \in \mathbb{R}^{4d_s}$.

C.3 Parallel state updates

Once the layer inputs and lower-layer hidden sequence are available, the candidates and gates can be computed independently of the current layer's cell recurrence. Fixing a memory layer and omitting its layer index, the cell update for head j in Eq. 7 has the affine form

$$\begin{aligned} C_t^{[j]} &= A_t^{[j]} C_{t-1}^{[j]} + B_t^{[j]}, \\ A_t^{[j]} &= \operatorname{diag}(f_t^{[j]}), \qquad B_t^{[j]} = \operatorname{diag}(i_t^{[j]}) e_t^{[j]} (k_t^{\text{mem},[j]})^\top. \end{aligned}$$

Affine updates compose associatively:

$$(A_2, B_2) \circ (A_1, B_1) = (A_2 A_1, A_2 B_1 + B_2).$$

A parallel prefix scan (Blelloch, 1990) computes the cumulative transformations, which are applied to the incoming cell state. The hidden sequence then follows from Eq. 8. We implement the scan in Triton (Tillet et al., 2019). Scanning is confined to a layer and a history block; block-end feedback determines the boundary state for the next block.

C.4 Implementation of block-end feedback

For each history block containing valid positions, $\mathcal{I}_b$ contains at most 32 distinct, evenly spaced valid positions shared across layers. For each sequence, let T_b be the last valid token position in history block b. Superscripts bef and aft denote states before and after feedback. The pre-feedback states and cached gates are

$$\begin{aligned} (C_b^{(\ell),\text{bef}}, D_b^{(\ell),\text{bef}}) &= (C_{T_b}^{(\ell)}, D_{T_b}^{(\ell)}), \\ (o_b^{(\ell)}, g_b^{(\ell)}) &= (o_{T_b}^{(\ell)}, g_{T_b}^{(\ell)}), \end{aligned}$$

where the states on the right are evaluated immediately before feedback. Empty blocks are skipped. All reconstruction losses and inner partial derivatives are evaluated at the same pre-feedback states.

In the following equations, a gate vector is reshaped to $n_h \times r \times 1$ and broadcast over the last matrix dimension. Thus, for a gate g and a state-shaped tensor A,

$$[g \odot A]_{j,a,c} = g_{j,a} A_{j,a,c}.$$

This is equivalent to left multiplication by $\operatorname{diag}(g^{[j]})$ within each head. Both $\tanh$ and its square act elementwise, and 1 in Eq. 11 denotes the all-ones tensor.

The compact gradient in Eq. 10 follows only the explicit depth-state connection in Eq. 8. For this derivative, sampled memory keys, values, and terminal gates are fixed. This defines the inner state correction and does not detach these quantities from the outer training graph when history gradients

are retained. Let $R_b^{(\ell)} = \partial \mathcal{L}_b^{(\ell)} / \partial D_b^{(\ell),\mathrm{bef}}$. The contributions from all strictly higher layers can be accumulated as

$$\begin{aligned} U_b^{(L)} &= 0, \\ U_b^{(\ell)} &= g_b^{(\ell+1)} \odot \left(R_b^{(\ell+1)} + U_b^{(\ell+1)} \right), \qquad \ell < L, \\ G_b^{(\ell)} &= o_b^{(\ell)} \odot \left(1 - \tanh^2\left(C_b^{(\ell),\mathrm{bef}} \right) \right) \odot U_b^{(\ell)}. \end{aligned} \tag{11}$$

Here $o_b^{(\ell)}$ and $g_b^{(\ell)}$ are the cached terminal gates. This recurrence is the chain-rule expansion of the strictly higher-layer loss gradient in Eq. 10; it excludes the layer's own reconstruction loss.

The candidate correction is computed separately for each memory head, with $\epsilon = 10^{-12}$ and $\eta = 0.01$:

$$\begin{aligned} a_b^{(\ell),[j]} &= \mathrm{RMS}\left(G_b^{(\ell),[j]} \right), \\ s_b^{(\ell),[j]} &= \max\left(1, \mathrm{RMS}\left(C_b^{(\ell),\mathrm{bef},[j]} \right) \right), \\ \widehat{\Delta C}_b^{(\ell),[j]} &= \begin{cases} -\eta s_b^{(\ell),[j]} G_b^{(\ell),[j]} / a_b^{(\ell),[j]}, & a_b^{(\ell),[j]} \geq \epsilon, \\ \mathbf{0}, & a_b^{(\ell),[j]} < \epsilon. \end{cases} \end{aligned} \tag{12}$$

RMS is computed over the $r \times r$ entries of a head, with correction arithmetic in FP32. In the following acceptance rule, we suppress the block, layer, and head indices. If the relative RMS of $\widehat{\Delta C}$ exceeds $\rho = 0.05$, we proportionally rescale it to ρ. We then form the candidate cell C^{cand} and cast it to the original state dtype. The candidate is accepted only if

$$\frac{\mathrm{RMS}\big(\mathrm{fp32}(C^{\mathrm{cand}}) - \mathrm{fp32}(C^{\mathrm{bef}})\big)}{s} \leq \rho + 10^{-7}.$$

Here fp32 denotes conversion to FP32. Otherwise, the original cell is retained. Here $\Delta C = C^{\mathrm{aft}} - C^{\mathrm{bef}}$ denotes the accepted change, including dtype rounding. Inactive heads retain their original states, and all accepted changes are committed simultaneously using Eq. 10. Nonfinite feedback losses, gradients, or corrected states abort the feedback step.

Using the cached terminal gates, expressed states are then rebuilt from shallow to deep layers:

$$\begin{aligned} D_b^{(\ell),\mathrm{aft}} = D_b^{(\ell),\mathrm{bef}} &+ o_b^{(\ell)} \odot \left[\tanh\left(C_b^{(\ell),\mathrm{aft}} \right) - \tanh\left(C_b^{(\ell),\mathrm{bef}} \right) \right] \\ &+ g_b^{(\ell)} \odot \left[D_b^{(\ell-1),\mathrm{aft}} - D_b^{(\ell-1),\mathrm{bef}} \right]. \end{aligned} \tag{13}$$

For the first memory layer, we set $D_b^{(0),\mathrm{bef}} = D_b^{(0),\mathrm{aft}} = \mathbf{0}$, so the lower-layer increment is zero. The incremental form preserves a state if neither its cell nor its lower-layer input changes. The highest-layer cell is unchanged because $U_b^{(L)} = 0$, while its expressed state may change through the corrected states below. The next history block starts from $(C_b^{(\ell),\mathrm{aft}}, D_b^{(\ell),\mathrm{aft}})$. Feedback leaves model parameters and outputs already computed in block b unchanged.

C.5 Training and initialization

Episode and objective. Each training episode starts from $C_0^{(\ell)} = D_0^{(\ell)} = \mathbf{0}$ at every memory layer. Available history is written into a memory stack $\mathcal{M}$, and the final assistant turn is predicted with writing and feedback disabled. For M supervised answer tokens $y_1, \ldots, y_M$, let $y_{<m} = (y_1, \ldots, y_{m-1})$. The objective is

$$\mathcal{L}_{\mathrm{ans}} = -\frac{1}{M} \sum_{m=1}^{M} \log p_{\phi,\theta}(y_m \mid y_{<m}, \mathrm{prompt}, \mathcal{M}). \tag{14}$$

Only adapter parameters θ are trained, while all backbone parameters ϕ remain frozen. Answers start without historical KV cache, with a 512-token read-sequence limit and context-dropout probability 0.2. Equation 9 defines the reconstruction objective used for the inner state correction. This objective is not added to the answer loss.

Long dataset. The second training stage uses the Long dataset, a synthetic collection of long-horizon memory episodes that we construct with a large language model. Each history is generated through an API in successive chunks that are concatenated into one long context, and each context is paired with several questions whose answers depend on it. We generate 100 candidate contexts in this way, using 1,350 chunk requests in total. Automatic checks reject 29 contexts, for example because they contain email addresses outside example domains, leak conversation-role markers, or include a question that reveals its answer. The remaining 71 contexts provide 832 question–answer pairs. Generation targets histories of about 32K and 64K tokens, but the accepted histories run longer, with 36 contexts in the shorter group and 35 in the longer group, spanning 44,074 to 98,340 tokens under the Qwen3 tokenizer. Each training example writes one full history into memory and then asks one of its questions, whose answer contains 6 to 15 tokens. Because answering starts without the historical KV cache, the model must recover each answer from memory, so these episodes train it to read information retained over long histories.

History gradients. QASPER (Dasigi et al., 2021) histories are capped at 8,192 tokens and retain gradients, allowing answer cross-entropy to differentiate through both memory updates and feedback correction. Fixing keys, values and gates for the inner partial derivative does not detach them from this outer training graph. Histories in the Long dataset are capped at 131,072 tokens, so none is truncated, and are written in 2,048-token blocks without gradients, with states detached between blocks. Their answer passes train memory reading and attention correction.

Training schedule. We train a fresh adapter on 2,240 QASPER presentations for 70 updates at 2×10^{-4}, then on the 832 examples of the Long dataset for 26 updates at 5×10^{-5} with a fresh optimizer. Both stages use seed 42, batch size 32 (eight workers, microbatch one, accumulation four), AdamW (Loshchilov & Hutter, 2017), BF16, ZeRO-2 (Rajbhandari et al., 2020), 10% warm-up and cosine decay. Feedback from all higher layers is enabled at history-block boundaries during training and inference. The feedback ablation disables feedback while retaining forward depth coupling, whereas the depth ablation sets the forward depth input $D_t^{(\ell-1)} = 0$. No auxiliary KL, teacher loss, write-sparsity penalty or additional forgetting stage is used.

Initialization. We initialize $W_{\text{pred}}^{\text{full}}$ and W_{read} separately to $[I_{d_s} \mid 0_{d_s\times d_s}]$, with zero biases. In the effective parameterization, $\widetilde{W}_{\text{pred}} = 0$, so initially $e_t = v_t^{\text{mem}}$ and read fusion selects temporal memory. All gate weights, including $\widetilde{W}_{\text{gate}}$, start at zero, with i, f, o, g biases $-1, +2, 0, -1$. Memory projections and W_x use Kaiming-uniform initialization, with $b_x = 0$. Correction projections use the first eight columns of the corresponding backbone query/output weights, normalized column-wise and scaled by 0.05, with remaining columns initialized to zero.

Online use. During inference, history blocks update and refine the carried C/D states with model parameters fixed. Memory readout continues during answer generation; answer-time state updates follow the corresponding evaluation protocol. Evaluation prompts retain the history specified by their protocols, and memory is reset for independent conversations.

Memory state during evaluation. We evaluate LSTMem with the prompt templates and memory-state handling of Lei et al. (2026) left unchanged, and our δ-Mem run after the second stage (Table 6) uses the same evaluation code, so the two runs differ only in the memory module. The memory is reset before every independent history, that is, before each MemoryAgentBench context, each LoCoMo conversation and each HotpotQA example, and every IFEval and GPQA-Diamond prompt starts from an empty memory. The history is written in 2,048-token blocks, and block-end feedback follows every block, including the last one, before the question is processed. The prompt also retains the history in the official format of each benchmark, so when answering, the backbone attends to the historical KV cache while the memory adds its attention corrections. The question and the generated answer read the memory and update it token by token as in δ-Mem, whereas block-end feedback is applied only to history blocks. All questions on the same history, including those of one LoCoMo conversation, start from the same memory state and historical KV cache obtained after the last history block, so updates made while answering one question never carry over to the next, and scores do not depend on question order.

D INFERENCE EFFICIENCY

LSTMem preserves most of δ-Mem's inference efficiency, both for a single decoding step and for the complete pipeline from history processing to answer generation, with a modest incremental cost for its richer memory interface. We compare the two methods on Qwen3-4B-Instruct, using the main configuration of LSTMem.

Decoding speed. Both methods use batch size one, BF16, FlashAttention (Dao, 2024), and the same optimized inference runtime. We evaluate nine KV-cache lengths from 8K to 128K tokens, where K denotes 1,024 tokens. Each method runs independently on each of four NVIDIA H100 80GB GPUs, with model order alternated across devices. For every model, cache length, and GPU, we collect 20 timed repetitions after two warm-up repetitions, yielding 1,440 measurements in total. Each reported latency is the mean of the four per-GPU medians. We time one synchronized token-decoding forward pass from restored KV and memory states. For latency t in milliseconds, the plotted *equivalent decode speed* is $1000/t$ tokens/s. This metric isolates the decoding step, excluding model loading, prefill, KV copying, memory-state restoration, warm-up, compilation, and token selection. The resulting values characterize single-step decoding from an existing memory state.

End-to-end cost. Because decoding speed excludes history processing, we also time the complete inference pipeline. Each run starts from an empty memory state, writes a history of 8K, 32K, or 128K tokens in 2,048-token blocks, including the block-end feedback of LSTMem, and then prefills the question and generates a 2,048-token answer. As in our evaluation protocol, the history KV cache remains available when answering, and model loading and warm-up are excluded. Each configuration is timed 40 times per method on H100 80GB GPUs with batch size one, and we report the mean over all timings.

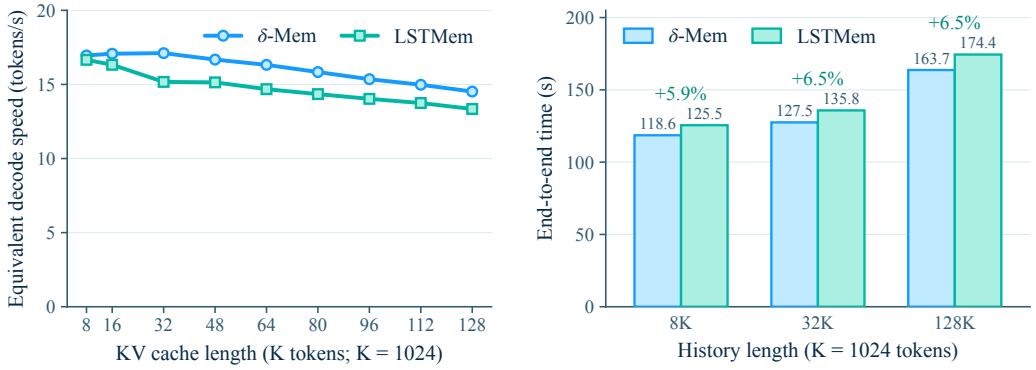

Figure 4: Inference efficiency of LSTMem and δ-Mem on Qwen3-4B-Instruct with batch size one on H100 80GB GPUs. (Left) Equivalent decode speed with KV caches from 8K to 128K tokens, given by the reciprocal of the mean per-GPU median single-token latency. (Right) Mean end-to-end time for processing an 8K, 32K, or 128K-token history and generating a 2,048-token answer. Labels give the relative increase of LSTMem over δ-Mem, computed from unrounded times.

Results. As shown in Figure 4 (left), LSTMem retains 92.10% of δ-Mem's equivalent decode speed on average, computed as the mean of the speed ratios across the nine cache lengths. The corresponding mean relative latency increase is 8.68%. At 128K tokens, LSTMem achieves 13.34 tokens/s versus 14.52 for δ-Mem, retaining 91.90% of its speed with only 6.07 ms of additional latency per step. The added latency remains below 7.5 ms at every evaluated length. With 2,048-token answers (Figure 4, right), LSTMem takes 125.5, 135.8, and 174.4 seconds for 8K, 32K, and 128K histories, compared with 118.6, 127.5, and 163.7 seconds for δ-Mem, so its end-to-end overhead is only 5.9% to 6.5%. Block-end feedback runs only while the history is processed, so its cost is amortized as the answer grows. These timings include diagnostic statistics computed during feedback that are not required for inference.