CONTINUAL CONTEXT COMPRESSION: TRAINING MODELS TO LEARN CONTINUALLY AT TEST TIME

Anonymous authors
Paper under double-blind review

ABSTRACT

We formalize test-time continual learning (TTCL), in which models continually acquire, retain, and use information from an open-ended deployment stream. We focus on observation-driven TTCL, where future queries are unknown during learning. The de facto solution, continual in-context learning (ICL), retains deployment history in context, causing memory and inference costs to grow with experience. We instead argue that models should be explicitly trained before deployment to learn continually at test time. Our training algorithm, Continual Context Compression (C3), combines long-context data, context distillation, and bounded memory to learn a state-update rule. The resulting algorithm performs test-time compression: it consolidates evicted context into persistent state in a single forward pass, without deployment-time supervision generation or gradient descent. Across repeated updates, it outperforms gradient-based sleep-time methods with orders of magnitude less deployment compute. Scaling to Qwen3-8B, C3 outperforms matched-budget memory baselines on future-use tasks, with strong acquisition and retention and limited interference with existing capabilities. Moreover, on held-out label-learning tasks, its state can even outperform continual ICL within the model's native context range. These results support TTCL as a first-class objective of model development; models should be trained not only for what they know, but for how effectively they learn afterward.

1 INTRODUCTION

Language models acquire most of their knowledge before deployment, yet operate in environments that continue to evolve. During deployment, they encounter new information about the world, which can shape their behavior through in-context learning (ICL). Unfortunately, context is a transient form of memory: once information leaves the context window, the model can no longer use it. This prevents experience from accumulating into persistent knowledge over time. Overcoming this limitation requires models to continually acquire and retain knowledge from an ongoing stream of experience—the central objective of continual learning (CL) (Thrun & Mitchell, 1995; Ring, 1997; Parisi et al., 2018; Delange et al., 2021).

Although continual learning has been studied extensively, prior work has largely focused on what we call *development-time continual learning*: models are repeatedly updated as new tasks or data arrive during model development, with the goal of acquiring new knowledge while preserving what the model has already learned (Kirkpatrick et al., 2017; Lopez-Paz & Ranzato, 2022; Shi et al., 2024). We instead study continual learning after deployment, during the phase of the model lifecycle conventionally referred to as test time. Unlike the development-time setting, there is no fixed boundary between learning and use: the model must continue to learn while it is being deployed. We refer to this problem as *test-time continual learning* (TTCL). Note that this differs from continual test-time adaptation (Wang et al., 2022), which typically only adapts a model to distribution shift to maintain performance on a fixed task.

Interest in learning during deployment is rapidly emerging across language modeling, memory, continual learning, and agentic systems (He et al., 2026; Wei et al., 2026; Hu et al., 2026; Yuksekgonul et al., 2026). These works instantiate different test-time learning problems rather than a shared setting. This fragmentation motivates a unified formulation of TTCL and the requirements a test-time continual algorithm should satisfy.

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.

We focus on a particularly relevant regime in which experience arrives before its future use is known: *observation-driven* TTCL. Models encounter information continuously but do not know in advance which observations will matter later. By contrast, many test-time learning settings reveal the future use through a known query, task, or repeated interaction (Yuksekgonul et al., 2026; Zhu et al., 2026; He et al., 2026). Observation-driven TTCL requires models to retain broadly useful knowledge from an open-ended stream without access to future queries.

We seek a continual algorithm that learns effectively in this setting. The de facto solution is continual ICL, which appends each new interaction to the running context. While general and simple, it grows without bound with the deployment stream. Scalable TTCL instead requires a repeatedly applicable update with bounded memory and practical deployment-time compute. Beyond scalability, the update should make past experience useful for future queries, continue to acquire new information without erasing earlier knowledge, and preserve the model's existing capabilities.

We argue that this ability to learn continually should itself be developed before deployment. We introduce *Continual Context Compression* (C3), a training algorithm that combines long-context data, context distillation, and a bounded memory architecture to equip a model with TTCL capabilities. The resulting continual algorithm performs *test-time compression*: as context is evicted, a learned forward-pass update consolidates it into persistent state. This contrasts with gradient-based sleep-time methods, which instead perform *sleep-time compression* by generating supervision and optimizing state during deployment (Eyuboglu et al., 2025; Behrouz et al., 2026). C3 invests in learning the update rule offline so that the deployed model can continually learn online with little additional compute.

Evaluating such an algorithm requires continual tasks that probe these capabilities. Following MemoryAgentBench (Hu et al., 2026), we adapt long-context and memory benchmarks to measure future use. We also devise metrics to measure acquisition and retention, and interference with existing capabilities. To our knowledge, our results provide the first demonstration of effective observation-driven TTCL. Our main technical contributions are:

- We formalize observation-driven TTCL through constraints and desired capabilities of continual algorithms, and instantiate continual evaluations that probe them.
- We introduce **Continual Context Compression** (C3), which trains a model to perform test-time compression through forward updates to a bounded state, without deployment-time data generation or gradient updates.
- We show that the learned algorithm achieves better continual performance than gradient-based sleep-time methods while using orders of magnitude less deployment-time compute.
- We scale C3 to Qwen3-8B, demonstrate broad gains over several memory baselines at matched budgets, and show that, on certain tasks, its learned state can even outperform continual ICL within the model's native context range.

2 Test-Time Continual Learning

Consider a model deployed in an environment over an indefinite time horizon. We write the interaction at time t as $e_t = (o_t, a_t, y_t)$, where o_t denotes information observed by the model, a_t the model's response or action, and y_t any subsequent outcome or feedback from the environment. At interaction t, the model carries forward a persistent state m_t, which conditions its response, and updates it from the current interaction e_t,

$$a_t \sim p_\theta(\cdot \mid m_t, o_t), \qquad m_{t+1} = U(m_t, e_t).$$

Here, θ denotes the model's initial parameters. As a notational convention, all persistent changes during deployment, including adapted weights, KV caches, and textual notes, are represented by m_t. We define *test-time continual learning* as learning through such persistent updates during deployment, allowing experience acquired earlier to influence behavior later.

We distinguish two sources of learning signal. In *observation-driven* learning, the model updates only from information it encounters, $m_{t+1} = U(m_t, o_t)$, before any query that will use it arrives. In *outcome-driven* learning, the update $m_{t+1} = U(m_t, o_t, a_t, y_t)$ may also use results from the model's behavior, such as rewards, errors, user corrections, or other environmental feedback. These sources are not mutually exclusive: realistic continual tasks may provide both observations to learn from and outcomes from which behavior can be improved.

In this work, we focus on observation-driven TTCL because it isolates the problem of continual memory from the additional challenges introduced by action-dependent feedback, such as exploration and credit assignment. Extending these capabilities to outcome-driven learning remains an important direction for future work.

2.1 Continual Algorithms for TTCL

A continual algorithm specifies the state and its update rule. For scalable observation-driven TTCL, we impose three constraints: **(i) Re-applicability:** the same update mechanism can be applied repeatedly without resetting the learner. **(ii) Bounded resources:** memory and per-interaction compute remain bounded as the deployment stream grows. **(iii) Deployment efficiency:** consolidation fits the workload's compute and latency budgets, including any supervision generation and optimization. A fixed-cost update that repeatedly stalls interaction or cannot keep pace with incoming observations is insufficient.

Within these constraints, we seek three capabilities: **(iv) Future-use generalization:** information acquired from the stream remains useful for queries and tasks that were not available when it was learned. **(v) Continual acquisition and retention:** the learner acquires new information while preserving useful information acquired earlier. **(vi) Non-interference:** continual learning preserves the model's existing parametric and in-context capabilities.

Existing approaches *retain*, *select*, or *compress* deployment history. Continual ICL retains the original experience but grows without bound and can struggle to use information in the middle of context (Liu et al., 2023). Fixed-budget selection, ranging from sliding-window policies such as StreamingLLM to content-aware KV eviction such as KeyDiff, achieves boundedness by discarding past information (Xiao et al., 2024; Park et al., 2026). Compression instead represents experience through textual summaries or external memory (Wang et al., 2025; Packer et al., 2024), or through bounded recurrent state in linear attention, state-space models, and hybrids with fixed-window attention (Gu & Dao, 2024; Dao & Gu, 2024; Yang et al., 2025b; Tumma et al., 2026). These architectures satisfy re-applicability and bounded resources by construction, but fixed capacity creates a tradeoff between acquiring new information and retaining earlier knowledge (Arora et al., 2023).

Learned consolidation offers another route: Cartridges optimizes compact KV representations through self-study, SHINE maps context to LoRA parameters in one forward pass, and Sleep consolidates experience through gradient-based updates (Eyuboglu et al., 2025; Liu et al., 2026; Behrouz et al., 2026). Such methods offer flexible representations, but have generally been studied in narrower settings or evaluated on only some of the requirements above. To our knowledge, no existing method has demonstrated an effective solution to observation-driven TTCL that satisfies these requirements jointly.

Under this formulation, compression can itself be a continual learning update: it incorporates experience into state that shapes future behavior, even after the original observations leave context. We argue that meeting these requirements should become an objective of model development. Our goal with C3 is to train a consolidation rule before deployment so that the resulting model can learn continually afterward through test-time compression (Section 3).

2.2 Continual Tasks and Evaluation

The desired capabilities determine what a continual evaluation must test. A *continual task* specifies the interaction stream, available learning signal, and evaluation.

Future-use generalization. Following MemoryAgentBench (Hu et al., 2026), we adapt long-context and memory benchmarks into streams whose evaluation queries are withheld during consolidation. We probe retrieval of previously observed facts, relational learning such as acquiring a label mapping from demonstrations, and distributional learning through aggregation over the stream. These require experience to support later uses that the update did not observe. Section 4 focuses on retrieval and test-time label learning.

Continual acquisition and retention. We measure how well the model learns new information and how much of that information it retains after later updates. We also report performance by evidence age to track forgetting over time.

Non-interference. We test whether the consolidated state interferes with information still available in the live context or with the model's existing capabilities. These controls separate gains from memory from damage to base-model performance. Together, these evaluations assess the quality of continual learning, while memory and compute measurements assess its deployment cost.

3 Continual Context Compression

C3 uses long-context data and context distillation to train a bounded-state forward update for observation-driven TTCL. We describe the deployment protocol, training objective, and write–merge–read architecture, then compare the resulting test-time compression algorithm with gradient-based sleep-time methods.

3.1 Training for Test-Time Compression

Problem formulation. Test-time compression maintains a bounded state from which a frozen model predicts as if it had access to the full deployment history. An *eviction policy* determines what remains in context; a *consolidation operator* incorporates evicted information into the state. We fix the eviction policy and learn the consolidation operator.

Blockwise deployment protocol. We use blockwise first-in-first-out (FIFO) eviction, retaining the most recent block of $B = 2048$ tokens (ablated in Appendix D). When a new block arrives, we evict the oldest block b_t and update the state as $m_{t+1} = C_\phi(m_t, b_t)$. Future predictions condition on the retained context and m_{t+1}. Both have fixed size, bounding total memory, while blockwise updates amortize consolidation over many tokens. The backbone parameters θ and compression parameters ϕ remain fixed during deployment.

Training objective. We train the consolidation operator through *context distillation* (Snell et al., 2022; Eyuboglu et al., 2025). The frozen backbone with access to the full history serves as the teacher, while the same backbone with bounded context and state serves as the student. For full history h_t and retained context c_t, we minimize

$$\mathcal{L}_{\mathrm{C3}} = \mathbb{E}\Big[\sum_t \sum_i \mathrm{KL}(p_\theta(\cdot \mid h_t, x_{<i}) \,\|\, p_{\theta,\phi}(\cdot \mid c_t, m_t, x_{<i}))\Big],$$

where t indexes consolidation cycles, i indexes prediction positions, θ denotes the frozen backbone, and ϕ the trainable compression parameters. Here, x is a continuation following cycle t, either further text or a question–answer pair about earlier content. The state m_t is obtained by unrolling C_ϕ over the preceding blocks, so gradients propagate through earlier consolidation cycles. No gradients flow through the teacher, and only ϕ is updated. This objective trains the bounded system to preserve the predictive behavior enabled by the missing context. Training streams must fit within the teacher's context window, but at deployment C_ϕ can be applied repeatedly beyond this horizon. On text continuations, we weigh each position of $\mathcal{L}_{\mathrm{C3}}$ by how strongly the evicted context changes the teacher's prediction—the KL between the teacher's full-history and retained-context distributions at that position—so that state capacity concentrates on predictions that depend on information no longer in context (Appendix D.1.4 ablates this weighting).

Separation of training objectives. C3 starts from a model that already predicts well with full context, then separately trains a compressor to preserve the behavior enabled by that context. By contrast, bounded-state architectures such as Mamba-2, Gated DeltaNet and TTT-E2E learn language modeling and compression jointly during pretraining (Dao & Gu, 2024; Yang et al., 2025b; Tandon et al., 2025). We choose the staged approach for trainability: full attention is already a strong substrate for learning language, whereas fixed-state recurrent models face a well-documented tradeoff between memory size and recall (Arora et al., 2023; 2025). We therefore first learn the predictive model without a memory bottleneck and only then train a bounded state to preserve its behavior. This gives compression an explicit target—the predictions of an already capable full-context model—rather than asking pretraining to jointly discover both how to model language and what information to retain.

Training data. Learning such a state requires a training signal that depends on information beyond the retained window. We construct long-context training examples from the 64K split of ProLong (Gao et al., 2025), truncate them to the context length supported by our Qwen3 backbones, and augment 20% of the data with question–answer examples following *Nemotron 3 Ultra* (NVIDIA et al., 2026).

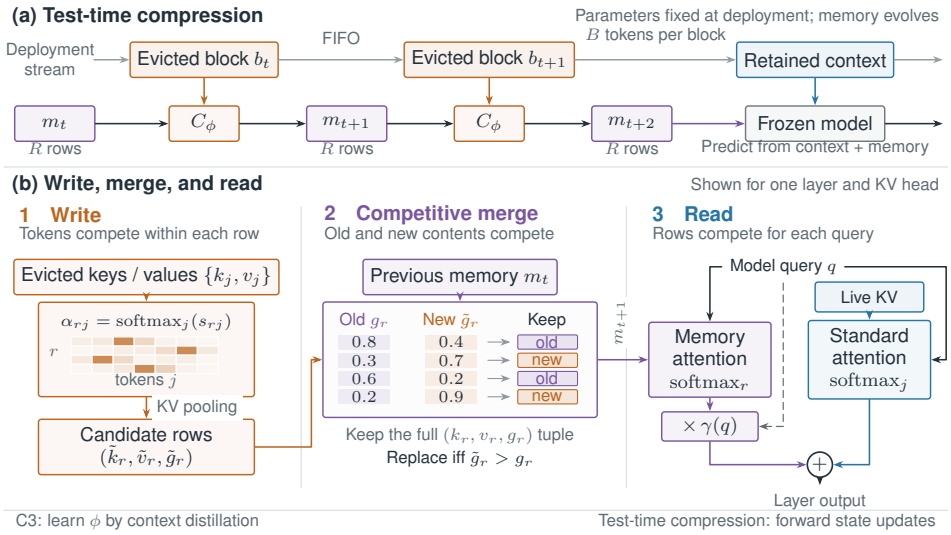

Figure 1: **Test-time compression learned with C3.** Evicted blocks update bounded memory through token pooling and competitive merging; a gated memory read augments attention.

Training ends with a short decay stage on a mixture of long-context text and continual task streams, including question answering, test-time label learning, and cross-block retrieval. In all cases, questions are introduced only after the relevant information has been observed, so the writer cannot condition its update on the future query. We describe data mixtures and training details in Appendix B.

3.2 Functional Form of the Test-Time Compressor

The learned compressor creates a three-timescale memory hierarchy. We interpret the frozen model parameters as *slow weights* and the live KV cache as *fast weights*: the former remain fixed throughout deployment, while the latter are written token-by-token and persist only within the context window. C3 introduces an intermediate state that is updated as context is evicted, but survives across interactions. In this sense, C3 extends the fast-weight programming view (Schlag et al., 2021) from two memory timescales to three. Its organizing principle is *competition*: tokens compete during writing, old and incoming contents compete during merging, and rows compete during reading. This limits the accumulation of interfering contributions characteristic of additive recurrent memories (Katharopoulos et al., 2020; Schlag et al., 2021; Arora et al., 2023). The write and merge together implement the consolidation operator C_ϕ.

Write. Suppressing layer and head indices, each token j in evicted block b_t receives a learned gate $g_j = \sigma(\mathrm{MLP}(d_j))$, where d_j describes its key, value and position. A learned row basis β_r attends over the block to form each candidate row,

$$s_{rj} = \beta_r^\top k_j/\tau + \log g_j,\ \alpha_{rj} = \mathrm{softmax}_j(s_{rj}),\ \tilde{k}_r = \sum_j \alpha_{rj} k_j,\ \tilde{v}_r = \sum_j \alpha_{rj} v_j,\ \tilde{g}_r = \sum_j \alpha_{rj} g_j.$$

Here τ is the write temperature. Token normalization makes each pooled row a convex combination of the block, avoiding norm growth from summation; the absorbed gate mass $\tilde{g}_r$ measures how strongly the block claims it.

Merge. A *competitive merge* selects the old or candidate row with larger absorbed mass,

$$(k'_r, v'_r, g'_r) = \begin{cases} (\tilde{k}_r, \tilde{v}_r, \tilde{g}_r), & \tilde{g}_r > g_r, \\ (k_r, v_r, g_r), & \text{otherwise.} \end{cases}$$

Selecting whole rows avoids blending their contents. The per-row maximum composes associatively across blocks; during training, hard selection uses a straight-through sigmoid relaxation of $(\tilde{g}_r - g_r)/\tau_m$, with temperature τ_m.

Read. Suppressing the per-layer read gain, each model query q attends over the rows through a separate softmax at temperature τ_r and a learned query-dependent gate $\gamma(q)$,

$$r(q) = \gamma(q) \sum_r \mathrm{softmax}_r\big(q^\top k'_r/\tau_r\big)\, v'_r.$$

This contribution is added alongside ordinary attention over retained context, so memory rows compete without taking attention mass from the live window. Figure 1 shows the write–merge–read path; implementation details are in Appendix B.

Ablation summary. Ablations under matched training budgets support token-normalized writing, competitive merging, and softmax reading for recovering evicted information. The read gate primarily protects in-window and parametric capabilities, with little effect on recovery. Appendix D reports the protocol, metrics, and architectural and objective ablations. We use this architecture and objective throughout, with additional projections and residual maps at 8B (Appendix B).

3.3 Amortized vs. Gradient-Based Sleep-Time Learning

A natural alternative to training a continual update rule beforehand is to optimize the model's state during deployment. Recent methods take this approach, using gradient descent on supervision constructed from the observed stream (Eyuboglu et al., 2025; Behrouz et al., 2026). Such updates can be performed either online or during *sleep time*, when interaction is paused and more compute can be spent. Sleep-time learning is attractive because it can tailor each update to the current deployment stream without requiring a learned continual updater. The cost is that data generation and optimization must be repeated throughout deployment.

C3 takes the opposite approach: it spends compute before deployment to learn the update rule, then performs test-time compression with a forward pass. Thus, we ask whether continual learning is better achieved by amortizing the update offline or by optimizing state during sleep time.

To isolate this choice, we compare the test-time compression algorithm learned by C3 with gradient-based sleep-time learning using the same frozen Qwen3-1.7B backbone, live window, and bounded KV rows. C3 uses its jointly trained writer and gated reader; the main GD baseline reads its rows through plain softmax. At each eviction, GD performs *self-study*: the model generates questions or requests about the evicted block while the full context is still available, obtains target answers from the full-context teacher, and then optimizes the state to reproduce those answers. We vary the number of self-study pairs at fixed optimizer steps. As a privileged reference, oracle GD instead receives the block's future test questions and gold answers, which neither C3 nor deployable self-study GD observes. Figure 2 reports continual NIAH results; Appendix C.1 gives the full protocol.

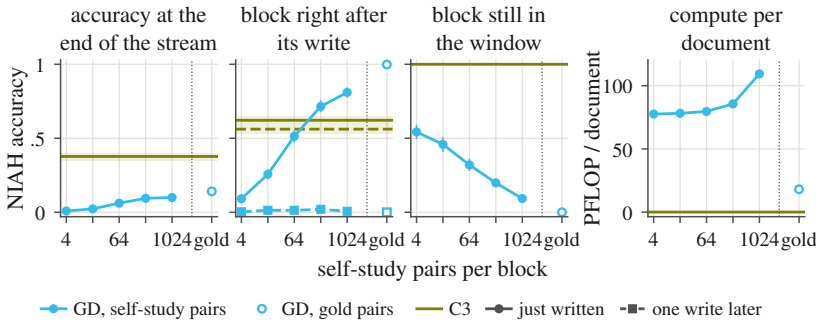

Figure 2: **Test-time versus sleep-time compression.** C3 and GD on continual NIAH with Qwen3-1.7B. From left: final accuracy; recall before (solid) and after (dashed) the next update; live-window accuracy; deployment compute, including supervision generation. The x-axis gives self-study pairs per block. Details: Appendix C.1.

Continual acquisition and retention. Gradient descent can fit newly observed information, but struggles to preserve it through subsequent updates. With enough self-study pairs, GD learns the newest block well, yet recall drops sharply after the next consolidation. Even oracle GD, which receives the actual future questions and gold answers, achieves near-perfect initial recall before forgetting. C3 sacrifices some immediate acquisition but retains substantially more of what it learns, and finishes the stream above every GD arm, including the oracle.

Non-interference. Gradient descent also interferes with information the model could already use. As self-study supervision increases, in-window recall declines, and oracle GD nearly destroys access to information still present in the live context. C3 largely preserves this capability. Giving GD C3's trained read gate reduces the interference, but also weakens acquisition and does not prevent

forgetting after the next update (Appendix C.3). Thus, the reader alone does not explain GD's failure under this protocol. Replay could improve retention, but a growing buffer violates bounded memory.

Deployment efficiency. Gradient descent requires substantial deployment compute to acquire useful information. Self-study must first generate supervision about each evicted block and then optimize the persistent state with repeated forward and backward passes; increasing the amount of supervision improves acquisition, but raises this cost further. Even oracle GD avoids only the generation step and remains expensive to optimize. C3 pays this cost once during training: at deployment, each block is consolidated with a single forward update, without generated supervision or backpropagation. In our comparison, C3 therefore achieves better continual performance while using orders of magnitude less deployment compute.

Supervision alignment. Even if deployment compute were free, self-study still has to guess what the model should learn. The generated questions and answers may not match the unknown future uses of the observed information, so more self-study does not necessarily produce better TTCL supervision. C3 avoids this problem by moving supervision design offline, where we can curate training streams and future-use objectives explicitly and optimize the update across repeated consolidations. This gives the learned updater a much broader training signal than the deployed model can reliably generate for itself.

Overall, these results show that **C3 improves both the quality and efficiency of continual learning by moving supervision design and update learning before deployment.** Appendix C provides further diagnostics of acquisition and retention across cycles, self-study and oracle supervision, and comparisons across different memory substrates and learned readers.

4 Results

We now scale C3 to Qwen3-8B (Yang et al., 2025a) and evaluate the resulting test-time compression algorithm across continual tasks. Re-applicability and bounded resources hold by construction; Section 3.3 studies deployment efficiency, and Figure 7 reports 8B consolidation time per update and per stream. Here we assess future-use generalization, continual acquisition and retention, and non-interference, before examining where the learned state exceeds full-history ICL. We use the recipe developed at Qwen3-1.7B, with training details in Appendix B.

4.1 Experimental setup

Anchors. Three references bracket the results. *Full-history ICL* is the continual ICL baseline and C3's teacher, retaining the whole history in context. *No-memory* keeps only the retention window. *Offline SFT* uses a LoRA shared across requests, trained on long-context web text and QA, retrieval, and labelling tasks, with neither TriviaQA nor TREC in the mixture. It separates gains from task exposure before deployment from learning the current stream (Appendix B).

Baselines. We match total persistent bytes, expressed in KV-cache token equivalents: the bytes of one cached token's keys and values across all layers. Each memory row stores a token-shaped key–value pair; C3 uses $r = 2048$ rows alongside the retained window unless stated otherwise. Continual baselines update at every consolidation. *KeyDiff* (Park et al., 2026) retains tokens whose keys are least similar to the average key. *Textual compaction* rewrites a running summary from previous notes and newly evicted context (Wang et al., 2025). Our fixed-budget adaptation of *Knowledge Seeding* (Behrouz et al., 2026) fits a per-request LoRA on self-written quizzes each cycle, paralleling the gradient-based sleep-time methods in Section 3.3. Its high consolidation cost limits our comparison to Qwen3-1.7B (Figure 12). For *SHINE* (Liu et al., 2026), a hypernetwork that writes context to LoRA parameters in one forward pass, we evaluate the authors' released checkpoint in its one-shot setting and train C3 with a matching state budget (Appendix E.1).

Evaluation. We follow MemoryAgentBench (Hu et al., 2026): documents arrive block by block, with evaluation questions withheld until the end, when most evidence has left the retention window. Accuracy pools evidence of all ages, including the live block. We hold memory fixed while extending streams within and beyond the backbone's native context range, using YaRN (Peng et al., 2026) beyond it. Appendix B.3 gives sample counts and dataset details; Appendix E provides complete task coverage and baseline comparisons.

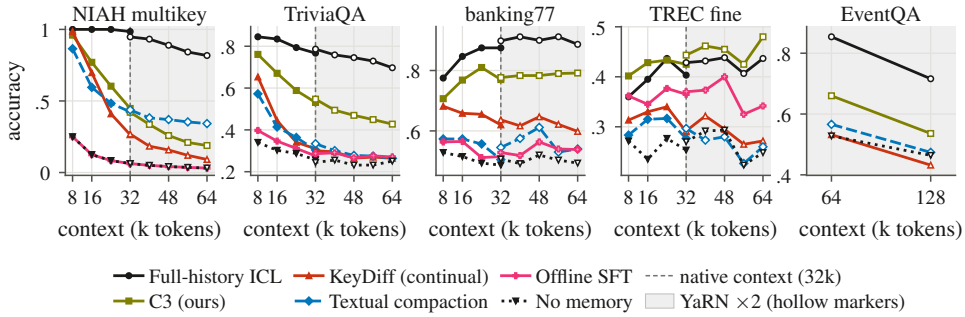

Figure 3: **Continual learning at matched memory budgets.** Qwen3-8B accuracy versus stream length. Dashed line: native context limit; hollow markers: YaRN, paired at the boundary. Details: Appendix B.3.

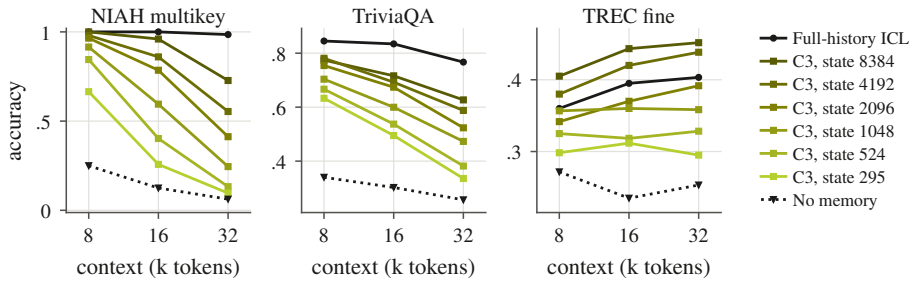

Figure 4: **State scaling analysis.** Qwen3-8B with C3 writers. Larger states improve accuracy across stream lengths. Legend values count KV-token equivalents: 2096 corresponds to $r = 2048$ rows.

4.2 Future-use generalization

We probe two MemoryAgentBench use cases. *Accurate retrieval* tests recall of facts and their compositions from novels (EventQA (Hu et al., 2026)), needles planted in each block (NIAH multikey (Hsieh et al., 2024)), and Wikipedia pages (TriviaQA (Joshi et al., 2017)). *Test-time learning* requires learning a label mapping from demonstrations and applying it to new inputs, using banking77 (Casanueva et al., 2020) and TREC-fine (Li & Roth, 2002). TREC is held out of every training mixture.

C3 leads the matched-budget baselines on EventQA and on the TriviaQA, banking77, and TREC-fine comparisons in Figure 3. On retrieval it remains below full-history ICL, while substantially improving over no memory. On banking77, accuracy grows with the stream as full-history ICL's does, while the baselines stay flat or decline. Since the retention window is fixed, demonstrations that have left it continue to teach the label mapping through the learned state. On TREC-fine, C3 can match or exceed full-history ICL; Section 4.5 examines this result. NIAH multikey is mixed: KeyDiff is strongest at short lengths, and textual compaction leads at extended lengths under YaRN.

Increasing state capacity improves recovery across the evaluated lengths on NIAH and TriviaQA and strengthens test-time label learning (Figure 4).

4.3 Continual acquisition and retention

The learner must acquire new information while keeping what it stored earlier. Figure 14 examines these requirements by the age of each question's evidence on fixed-length TriviaQA streams. Even the oldest block remains close to full-history ICL, and every evicted age improves over no memory. Recently evicted blocks, however, are recalled less accurately than the oldest ones.

This pattern points to an admission bottleneck rather than uniform decay with age. The competitive merge keeps the old or incoming content with the larger absorbed mass, so a newly evicted block must outbid rows already stored. Strong retention of earlier blocks can therefore come at the expense of admitting later ones. Increasing capacity improves recovery, consistent with reducing this competition (Figure 4).

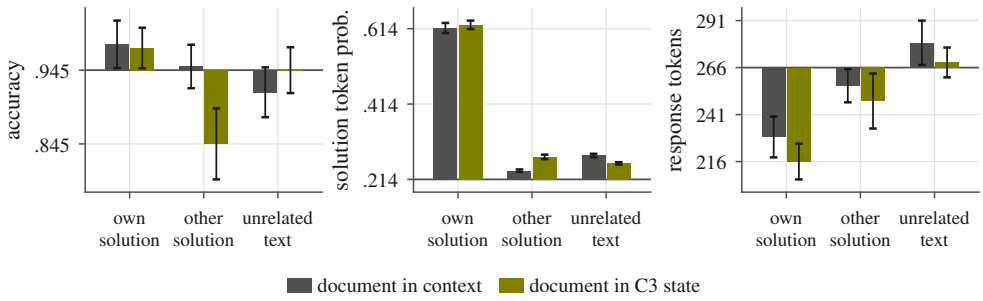

Figure 5: **Non-interference on GSM8K.** Changes from the no-document baseline with relevant or unrelated text in context or C3's state. Error bars are paired 95% intervals. We detail the experimental setting in Appendix B.3.

4.4 Non-interference

A consolidated state is a new source of information alongside the frozen weights and retained KV cache. Therefore, we explore whether it disrupts either capability. To probe parametric knowledge, we use GSM8K (Cobbe et al., 2021), whose problems can be answered without memory (Catalan-Tatjer & Geiping, 2026). Before each problem, we provide its own solution, another problem's solution, or unrelated text from Paul Graham essays, either in context or consolidated into the learned state. Figure 5 reports changes relative to the no-document baseline.

Parametric capabilities. The problem's own solution raises accuracy and solution-token probability through both routes, and answers become shorter. Unrelated text produces no clear accuracy change. Another problem's solution is largely harmless in context but reduces accuracy after consolidation. The comparison with SHINE and offline SFT shows considerably larger interference for those systems (Figure 15). Thus C3 substantially limits interference, though it does not eliminate every case.

In-window ICL. The age-zero point in Figure 14 evaluates questions whose evidence remains in the live window. It is apparent that C3 does not struggle to retrieve information rom this block, as it maintains high accuracy.

These demonstrate that a model can be trained to use C3's state while distinguishing persistent learning from disruption of its ordinary behaviour.

4.5 Beyond continual ICL

C3 trains the compressor to match full-history ICL, yet the learned state exceeds ICL on TREC-fine and TREC-coarse (Figure 11). TREC is absent from every training mixture, and each stream permutes its label IDs, so the advantage cannot come from a fixed label prior learned during training. It does not extend to banking77, where full-history ICL remains ahead. The advantage depends on state capacity and training. In Figure 4, the larger writers exceed full-history ICL across the native context range, while the smaller writer trained with a shorter final stage stays below it. Compression can therefore support continual learning beyond reproducing the teacher: on these tasks, the learned state supports new label predictions better than reading all the demonstrations directly, even when the full history fits in context.

5 Conclusion

Today's language models largely stop learning when training ends. This work points toward a different model lifecycle, where learning continues after deployment. We formalize this setting, test-time continual learning, and show that a model can be trained in advance to compress an ongoing stream into persistent knowledge through inexpensive forward updates. Continual Context Compression demonstrates that this approach can outperform both deployment-time optimization and conventional memory management baselines. In fact, in some settings it even learns representations that are more useful than the full interaction history itself. More broadly, our results argue for making continual learning an explicit training objective rather than a deployment-time add-on. This shifts the model lifecycle from train-then-freeze toward systems that continue to improve as they are used.

AI USE STATEMENT

We used generative AI tools for code assistance for implementing standard methods, data processing and filtering, and visualizing results. We additionally used them for editing the manuscript, which the authors reviewed and edited. The problem framing, experimental design, and interpretation of results are the authors' own. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models, 2023. URL https://arxiv.org/abs/2312.04927.

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff, 2025. URL https://arxiv.org/abs/2402.18668.

Ali Behrouz, Farnoosh Hashemi, Adel Javanmard, and Vahab Mirrokni. Language models need sleep: Learning to self-modify and consolidate memories, 2026. URL https://arxiv.org/abs/2606.03979.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline, 2020. URL https://arxiv.org/abs/2004.07211.

Iñigo Casanueva, Tadas Temčinas, Daniela Gerz, Matthew Henderson, and Ivan Vulić. Efficient intent detection with dual sentence encoders, 2020. URL https://arxiv.org/abs/2003.04807.

Albert Catalan-Tatjer and Jonas Geiping. Learning-forgetting optimality in supervised finetuning: A cliff perspective. In *High-dimensional Learning Dynamics 2026*, 2026. URL https://openreview.net/forum?id=aWl2TIUjxP.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-lora: Learning to instantly internalize contexts, 2026. URL https://arxiv.org/abs/2602.15902.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem, 2019. URL https://arxiv.org/abs/1812.00420.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality, 2024. URL https://arxiv.org/abs/2405.21060.

Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021. ISSN 1939-3539. doi: 10.1109/tpami.2021.3057446. URL http://dx.doi.org/10.1109/TPAMI.2021.3057446.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study, 2025. URL https://arxiv.org/abs/2506.06266.

Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. How to train long-context language models (effectively), 2025. URL https://arxiv.org/abs/2410.02660.

Shashwat Goel, Nikhil Chandak, Arvindh Arun, Ameya Prabhu, Steffen Staab, Moritz Hardt, Maksym Andriushchenko, and Jonas Geiping. Futuresim: Replaying world events to evaluate adaptive agents, 2026. URL https://arxiv.org/abs/2605.15188.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL https://arxiv.org/abs/2312.00752.

Yufei He, Juncheng Liu, Yue Liu, Yibo Li, Tri Cao, Zhiyuan Hu, Xinxing Xu, and Bryan Hooi. Evotest: Evolutionary test-time learning for self-improving agentic systems, 2026. URL https://arxiv.org/abs/2510.13220.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in llm agents via incremental multi-turn interactions, 2026. URL https://arxiv.org/abs/2507.05257.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension, 2017. URL https://arxiv.org/abs/1705.03551.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020. URL https://arxiv.org/abs/2006.16236.

Jang-Hyun Kim, Junyoung Yeom, Sangdoo Yun, and Hyun Oh Song. Compressed context memory for online language model interaction, 2024. URL https://arxiv.org/abs/2312.03414.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, March 2017. ISSN 1091-6490. doi: 10.1073/pnas.1611835114. URL http://dx.doi.org/10.1073/pnas.1611835114.

Xin Li and Dan Roth. Learning question classifiers. COLING '02, pp. 1–7, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1072228.1072378. URL https://doi.org/10.3115/1072228.1072378.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023. URL https://arxiv.org/abs/2307.03172.

Yewei Liu, Xiyuan Wang, Yansheng Mao, Yoav Gelbery, Haggai Maron, and Muhan Zhang. Shine: A scalable in-context hypernetwork for mapping context to lora in a single pass, 2026. URL https://arxiv.org/abs/2602.06358.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning, 2022. URL https://arxiv.org/abs/1706.08840.

Martin Marek, Dongkyu Cho, Shikai Qiu, Rumi Chunara, Pavel Izmailov, and Andrew Gordon Wilson. Forgetting in language models: Capacity, optimization, and self-generated replay, 2026. URL https://arxiv.org/abs/2605.26097.

NVIDIA, :, Aaron Blakeman, Aaron Thomas, Aastha Jhunjhunwala, Abhibha Gupta, Abhinav Khattar, Adam Rajfer, et al. Nemotron 3 ultra: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning, 2026. URL https://arxiv.org/abs/2606.15007.

Charles O'Neill, Alex Sandomirsky, Harry Partridge, Mudith Jayasekara, and Max Kirkby. Still: Amortized kv cache compaction in a single forward pass, 2026. URL https://arxiv.org/abs/2606.07878.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *CoRR*, abs/1802.07569, 2018. URL http://arxiv.org/abs/1802.07569.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023. URL https://arxiv.org/abs/2304.03442.

Junyoung Park, Dalton Jones, Matthew J Morse, Raghavv Goel, Mingu Lee, and Chris Lott. Keydiff: Key similarity-based kv cache eviction for long-context llm inference in resource-constrained environments, 2026. URL https://arxiv.org/abs/2504.15364.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models, 2026. URL https://arxiv.org/abs/2309.00071.

Mark Ring. Child: A first step towards continual learning. *Machine Learning*, 07 1997. doi: 10.1023/A:1007331723572.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers, 2021. URL https://arxiv.org/abs/2102.11174.

Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, and Hao Wang. Continual learning of large language models: A comprehensive survey, 2024. URL https://arxiv.org/abs/2404.16789.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context, 2022. URL https://arxiv.org/abs/2209.15189.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts, 2020. URL https://arxiv.org/abs/1909.13231.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory, 2025. URL https://arxiv.org/abs/2504.07952.

Arnuv Tandon, Karan Dalal, Xinhao Li, Daniel Koceja, Marcel Rød, Sam Buchanan, Xiaolong Wang, Jure Leskovec, Sanmi Koyejo, Tatsunori Hashimoto, Carlos Guestrin, Jed McCaleb, Yejin Choi, and Yu Sun. End-to-end test-time training for long context, 2025. URL https://arxiv.org/abs/2512.23675.

Sebastian Thrun and Tom M. Mitchell. Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1):25–46, 1995. ISSN 0921-8890. doi: https://doi.org/10.1016/0921-8890(95)00004-Y. URL https://www.sciencedirect.com/science/article/pii/092188909500004Y. The Biology and Technology of Intelligent Autonomous Agents.

Neehal Tumma, Noel Loo, and Daniela Rus. Preconditioned deltanet: Curvature-aware sequence modeling for linear recurrences, 2026. URL https://arxiv.org/abs/2604.21100.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization, 2021. URL https://arxiv.org/abs/2006.10726.

Qin Wang, Olga Fink, Luc Van Gool, and Dengxin Dai. Continual test-time domain adaptation, 2022. URL https://arxiv.org/abs/2203.13591.

Qingyue Wang, Yanhe Fu, Yanan Cao, Shuai Wang, Zhiliang Tian, and Liang Ding. Recursively summarizing enables long-term dialogue memory in large language models, 2025. URL https://arxiv.org/abs/2308.15022.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory, 2026. URL https://arxiv.org/abs/2511.20857.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL https://arxiv.org/abs/2309.17453.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule, 2025b. URL https://arxiv.org/abs/2412.06464.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length, 2025c. URL https://arxiv.org/abs/2406.06484.

Tianzhu Ye, Li Dong, Qingxiu Dong, Xun Wu, Shaohan Huang, and Furu Wei. Online experiential learning for language models, 2026. URL https://arxiv.org/abs/2603.16856.

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. Helmet: How to evaluate long-context language models effectively and thoroughly, 2025. URL https://arxiv.org/abs/2410.02694.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent, 2026. URL https://arxiv.org/abs/2507.02259.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time, 2026. URL https://arxiv.org/abs/2601.16175.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence, 2017. URL https://arxiv.org/abs/1703.04200.

Zheyuan Zhang, Zehao Wen, Alvin Zhang, Andrew Wang, Jianwen Xie, Daniel Khashabi, and Tianmin Shu. Agentodyssey: Open-ended long-horizon text game generation for test-time continual learning agents, 2026. URL https://arxiv.org/abs/2606.24893.

Xinyu Zhu, Zhe Xu, Xiaohan Wei, Yunchen Pu, Fei Tian, Chonglin Sun, Kaushik Rangadurai, Hua Zhi, Frank Shyu, Sandeep Pandey, Luke Simon, Yu Meng, and Xi Liu. Self-guided test-time training for long-context llms, 2026. URL https://arxiv.org/abs/2607.09415.

A Extended Related Work

Development-time continual learning and test-time learning. Classical continual learning studies how a model acquires new information while preserving earlier knowledge (Parisi et al., 2018; Delange et al., 2021). Replay-based methods revisit earlier examples or constraints (Lopez-Paz & Ranzato, 2022; Chaudhry et al., 2019; Buzzega et al., 2020), while regularization-based methods limit changes to parameters important for previous tasks, as in EWC and Synaptic Intelligence (Kirkpatrick et al., 2017; Zenke et al., 2017). In language models, this often appears as continual pretraining or fine-tuning as new data or tasks arrive (Shi et al., 2024; Marek et al., 2026). These methods address the same acquisition–retention tension as TTCL, but place learning within model development rather than deployment.

A separate literature moves learning into test time. Test-time training adapts a model from the current test input or stream (Sun et al., 2020; Wang et al., 2021), while continual test-time adaptation targets non-stationary deployment distributions, typically for a fixed downstream task (Wang et al., 2022). More recent work moves toward continual deployment-time learning: Online Experiential Learning consolidates knowledge from deployment trajectories (Ye et al., 2026), FutureSim evaluates adaptation as real-world information arrives (Goel et al., 2026), and EvoTest and Dynamic Cheatsheet accumulate reusable experience across problems (He et al., 2026; Suzgun et al., 2025). These works instantiate different task assumptions, learning signals, and persistence mechanisms. Our TTCL formulation instead treats continual learning during deployment as the object of study and explicitly requires repeated learning over an indefinite horizon. AgentOdyssey similarly uses the language of test-time continual learning for interactive agents (Zhang et al., 2026), but focuses on outcome-driven learning through exploration and environment interaction; we study the complementary observation-driven setting in which future use is unknown when information is acquired.

Long-context and bounded-state sequence models. Long-context modeling and TTCL both require information from the distant past to influence future predictions. Extending attention preserves direct access to more history, while bounded-state architectures summarize it. Linear attention (Katharopoulos et al., 2020), state-space models such as Mamba and Mamba-2 (Gu & Dao, 2024; Dao & Gu, 2024), and gated recurrent architectures such as Gated DeltaNet (Yang et al., 2025b) maintain recurrent states whose memory does not grow with sequence length. These architectures therefore satisfy important structural requirements of a continual algorithm, although fixed-capacity state creates a tradeoff between memory size and associative recall (Arora et al., 2023). Long-context evaluations such as HELMET further show that synthetic retrieval alone does not capture the full range of long-context capabilities (Yen et al., 2025).

Test-time-training architectures provide another bounded-state approach. TTT-E2E meta-trains a model so that gradient updates on incoming context improve later prediction (Tandon et al., 2025). C3 shares the principle that later learning should be prepared in advance, but separates language modeling from continual compression. Whereas bounded-state recurrent models typically learn both jointly during pretraining, C3 starts from a full-context Transformer and trains a second-stage compressor against its predictions. This gives compression an explicit teacher rather than requiring language modeling and selective retention to emerge through the same memory bottleneck. Jointly learning these objectives during pretraining remains an important direction for future work.

Memory agents and persistent memory. A large literature gives language-model agents persistent memory outside the standard context window. Generative Agents and Reflexion store and reuse past experience (Park et al., 2023; Shinn et al., 2023); MemGPT manages multiple memory tiers (Packer et al., 2024); Dynamic Cheatsheet maintains a self-curated library of reusable strategies (Suzgun et al., 2025); and Evo-Memory studies memories that evolve across sequential task streams (Wei et al., 2026). MemAgent instead trains a memory policy with reinforcement learning to process arbitrarily long inputs through a fixed context window (Yu et al., 2026).

These systems treat memory as an agent policy, textual store, or task-specific long-context mechanism. C3 instead trains a bounded internal state specifically for repeated observation-driven learning. External memories can also implement TTCL under our definition, but must satisfy the same requirements on bounded resources, future-use generalization, acquisition and retention, and non-interference. MemoryAgentBench makes this connection explicit by evaluating retrieval, test-time learning, long-range understanding, and selective forgetting under incremental interaction (Hu et al., 2026); we build on its streaming perspective.

Context compression and state internalization. The closest methods compress context into a smaller learned representation. Compressed Context Memory (CCM) (Kim et al., 2024) recursively compresses an expanding interaction history into KV memory, including a fixed-size variant that averages successive compressed states. CCM shows that recurrent compression can preserve downstream performance while reducing memory, including in streaming and unseen-task settings. Its focus, however, is context efficiency rather than continual learning: it does not separately evaluate acquisition, retention across later updates, or interference with existing capabilities. C3 instead trains a fixed-capacity state for repeated observation-driven learning and evaluates these behaviors directly; architecturally, it replaces CCM's averaging update with content-dependent competition between incoming and stored information.

Still (O'Neill et al., 2026) similarly amortizes KV-cache compaction into a learned forward pass that can be applied iteratively as context grows. Still emphasizes the speed–quality tradeoff of long-context compaction, whereas C3 trains and evaluates the state as a continual learner, including future-use generalization, acquisition and retention, and non-interference.

Other forward methods internalize context into model state. SHINE (Liu et al., 2026) maps context to LoRA parameters in one forward pass, while Doc-to-LoRA (Charakorn et al., 2026) similarly generates context-specific adapters. SHINE-R processes chunks sequentially but retains the resulting adapters, so persistent memory grows with the stream. C3 instead repeatedly updates a fixed-capacity state. These methods nevertheless share the broader principle that offline training can amortize the cost of adapting to future context.

Gradient-based methods instead perform context-specific learning during deployment. Cartridges (Eyuboglu et al., 2025) fits compact KV states using self-generated study examples and context distillation, while sleep-based consolidation (Behrouz et al., 2026) transfers deployment experience into persistent parameters through optimization and rehearsal. We refer to this family as *sleep-time compression*: supervision is generated and state is optimized after deployment begins. C3 learns the update rule offline, so deployment requires only forward *test-time compression*. Section 3.3 compares these strategies directly; in our continual setting, the tested gradient-based methods require substantially more deployment compute while struggling with supervision coverage, retention across updates, and interference.

B Experimental details

B.1 Backbones and token equivalence

We use Qwen3-1.7B and Qwen3-8B (Yang et al., 2025a), retaining three sink tokens and the newest 2048-token block. We measure state in *KV-token equivalents*: the bytes needed to cache one token's keys and values across all layers. Budgets include keys, values, and absorbed gate masses. At 8B, each C3 row stores a BF16 key and value per layer and KV head, plus its gate mass, for a total of 150,912 bytes. Thus r rows occupy approximately $1.023r$ equivalents: 2048 rows occupy 2096, and 8192 occupy 8384. The live window occupies 2051 equivalents on both backbones. Figure 6 reports C3's trainable parameters at each 8B state size; these shared parameters remain fixed at deployment and are separate from the per-stream state. The 8B writers also learn three maps not shown in Section 3.2: a per-layer projection of the keys used in the write scores s_{rj}, a residual map on the stored values ($v_j + W_v v_j$), and a residual ReLU MLP on each pooled row's key and value. The projection is initialized to the identity and the two residuals to zero, so training starts from the write equations as stated.

All decoding is greedy, with thinking disabled. Unless noted below, beyond the native 32K context range all methods, including full-history ICL, use static YaRN (Peng et al., 2026): factor 2 up to 64K tokens and factor 4 up to 164K. The EventQA exception is specified below.

B.2 Training data and recipe

Data. We use the 64K split of ProLong (Gao et al., 2025), with 20% question–answer augmentation following the Nemotron-3-Ultra recipe (NVIDIA et al., 2026). The training dataset contains 6.25B tokens. The final decay stage mixes ProLong text with continual task streams. At 1.7B, the sampling weights are 50% text, 15% SQuAD, 10% HotpotQA, 15% test-time label learning, 5% multi-query

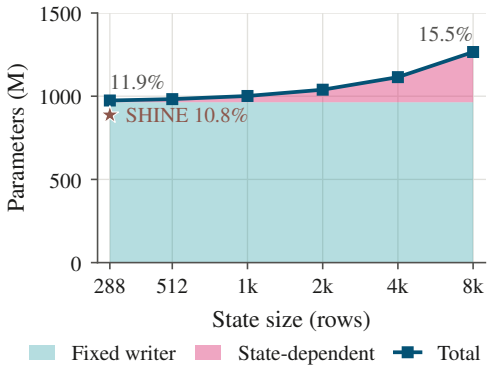

Figure 6: **Trainable parameters versus state size on Qwen3-8B.** C3 combines state-independent writer parameters with one learned 128-dimensional key per row, KV head, and layer. Percentages are relative to the 8.19B backbone. The star marks SHINE (Liu et al., 2026) at the matching 288-row C3 state size. SHINE's hypernetwork, meta-LoRA, and memory tokens total 887.6M trainable parameters and produce a rank-8 LoRA occupying 296 KV-token equivalents. Only its memory tokens and position embeddings (1.2M parameters) scale with adapter rank.

NIAH, and 5% variable tracking. At 8B, they are 10% text, 60% SQuAD, 20% test-time label learning, 5% multi-query NIAH, and 5% variable tracking. TriviaQA and TREC are excluded from training.

Objective and optimization. We train by context distillation from the frozen full-history ICL teacher, using KL temperature $\tau_{\mathrm{KL}} = 2$ (Hinton et al., 2015) and gradients through every consolidation. On text continuations, the loss at each position is weighted by the KL divergence between the teacher's full-history and retained-context predictions, emphasizing positions that depend on evicted information. We use AdamW with learning rate 10^{-3}, 50 warmup steps, and 48 streams per optimizer step.

Training schedule. A warmup–stable–decay schedule lets us continue one training *trunk* per state size without fixing its training horizon. To reduce ablation cost, the 1.7B writers use the same objective and schedule on 16.5K-token streams.

B.3 Evaluation protocols and metrics

Streaming protocol. We follow MemoryAgentBench (Hu et al., 2026). A stream contains $c+1$ blocks of 2048 tokens and undergoes c consolidations. Blocks come from independent documents except in the whole-document tasks, EventQA and En.Sum. Evaluation questions arrive only after the final consolidation, so the writer cannot condition its updates on them. Unless noted otherwise, we average accuracy over the questions of 50 streams per context length. Age-conditioned results distinguish evidence in the live window (age zero) from evicted evidence (age at least one).

NIAH multikey. Following RULER, each block contains four needles in a Paul Graham essay haystack, with keys drawn from a vocabulary split excluded from training and 7-digit values. We query one key per block and score an answer as correct when it contains every required answer string.

TriviaQA. Each block contains the first 2048 tokens of a Wikipedia evidence page. We use that page's questions, retaining those the backbone answers with the page but not without it, and score answers by normalized alias containment.

Test-time label learning. Each block contains about 120 labelled demonstrations. After the stream, we ask 12 held-out questions from the official evaluation split, prompt the answer after `label:`, and score the first integer by exact match. Banking77 has 77 labels and retains their official IDs. TREC-fine has 50 labels, permutes their IDs per stream, and is excluded from all training mixtures.

EventQA. We stream each of five novels in full and then ask its 100 six-way next-event questions. Streams contain 33 blocks in the 64K setting and 65 in the 131K setting. At 64K, all methods use YaRN factor 2; at 131K, the baselines use factor 4 and C3 uses factor 2.

En.Sum. We stream ten InfiniteBench novels of 81K–156K tokens using YaRN factor 4, then request a summary and score it by ROUGE-L-Sum against the reference. In Figure 11, EventQA and En.Sum scores are plotted against document length; the EventQA results are the same as in Figure 3.

GSM8K. The non-interference evaluation (Section 4.4) uses 200 test items. Before each problem, we provide its own solution, another problem's solution, or unrelated Paul Graham text, either in the live window or consolidated into state. The no-document condition serves as the reference. We report accuracy, solution-token probability, and response length, with paired 95% intervals.

Additional metrics. We use token F1 for NarrativeQA and QASPER, multiple-choice accuracy for LongHealth, and teacher KL where indicated. Appendix D.1 defines the closure metric used in the ablations.

B.4 Baseline settings

Reference conditions and budgets. Full-history ICL retains the entire stream; no memory retains only the 2051-token live window. At C3's default state size, matched-budget selection and textual compaction receive an additional 2096 KV-token equivalents. All per-stream state counts toward this budget, except the explicitly excluded Knowledge Seeding replay buffer below. Parameters shared across streams are accounted for separately.

KeyDiff. The 8B comparison retains 2051 live tokens plus 2096 selected tokens. At each consolidation, KeyDiff re-selects the latter by one minus the cosine similarity between each token's key and the mean key direction, averaged over layers and heads. The newest block is retained without scoring.

Textual compaction. At every eviction, the model rewrites its running notes from the previous notes and the evicted block, with instructions to preserve names, numbers, and identifiers. Notes are capped at 2096 tokens and supplied as text alongside the retained context.

Offline SFT. We train a static rank-16 LoRA on the MLP projections for 1000 next-token cross-entropy steps at learning rate 3×10^{-4}, using 16.5K-token streams, the same eviction policy, and the 1.7B decay mixture. The adapter is shared across streams and remains fixed at deployment, when the model uses plain eviction.

Knowledge Seeding. This baseline fits a per-stream LoRA on every layer's MLP projections, counting both adapter factors toward the state budget. We use rank 175 at 1.7B (2100 equivalents; Figure 12) and rank 87 at 8B (2088 equivalents; Figure 7). At each eviction, self-study processes 512-token chunks of the evicted block, with 32 generations and one teacher-written summary pair per chunk. Starting from the previous cycle's adapter, it takes 200 Adam steps at learning rate 3×10^{-5} to minimize KL to the full-context teacher. Half the loss uses student-sampled continuations, the on-policy term of generalized distillation.

Compared with the original method (Behrouz et al., 2026), we omit parameter expansion to enforce bounded memory and omit its RL-based imitation loss. We also omit the separate dreaming stage, which targets self-improvement beyond the observation-driven memory consolidation studied here.

The continual variant uses only the newest block's pairs; replay uses all pairs generated so far, with its quiz buffer excluded from the reported adapter budget. A learning rate of 10^{-3} damaged in-window reading and is not reported. Deployment cost limits the continual performance comparison to 1.7B; the 8B setting is used only for timing. The controlled row-state GD experiment in Section 3.3 is a separate baseline, detailed in Appendix C.

SHINE. We use the authors' released Qwen3-8B checkpoint in its one-shot setting. The hypernetwork maps all context preceding the newest block to a rank-8 LoRA on the attention and MLP projections in one forward pass (296 equivalents). It answers using C3's prompt and the same retained block. We train a C3 state with a matching memory budget for this comparison.

B.5 Compute and hardware

Deployment compute for gradient-based sleep-time methods includes supervision generation and state optimization. C3's offline training cost is accounted for separately from its deployment-time forward updates.

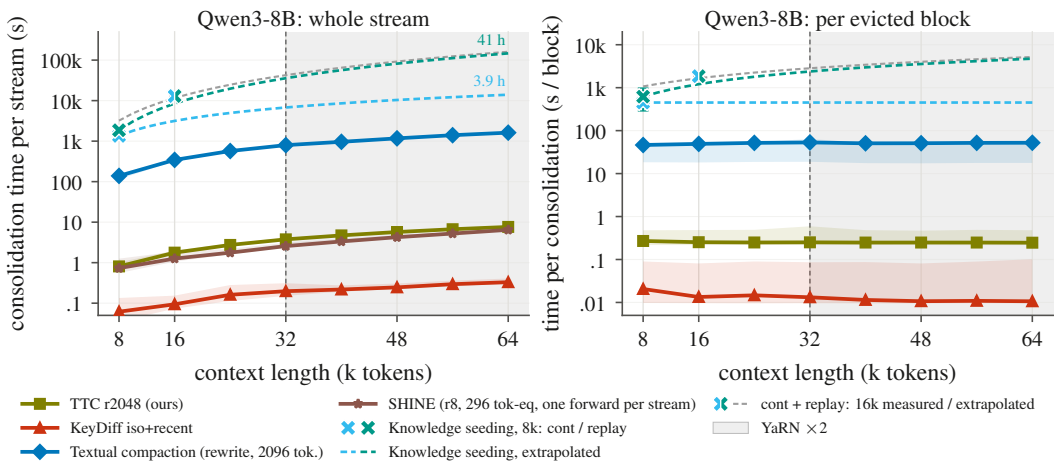

Figure 7: **Consolidation time on Qwen3-8B.** NIAH-single streams of 2048-token blocks, measured on one NVIDIA H100 80GB. Left: total consolidation time per stream. Right: time per consolidation. Full-history ICL, no memory, and offline SFT perform no consolidation and are omitted.

Training uses AMD MI325X nodes. A trunk run uses 48 GPUs at about 52 s per step, or roughly 2800 GPU-hours for 4000 steps; a decay run uses eight GPUs for about four hours. Evaluation uses NVIDIA A100, H100, and B200 GPUs. Figure 7 reports consolidation time per evicted block on one H100 per system.

C TEST-TIME VS. SLEEP-TIME COMPRESSION

We extend Section 3.3 with diagnostics of acquisition and retention across cycles, the information learned from self-study supervision, and comparisons across memory substrates and readers.

C.1 PERFORMANCE OVER CYCLES

Protocol. We use frozen Qwen3-1.7B on 100 documents per task, streamed as eight 2048-token blocks. The main comparison (Figure 2) uses NIAH multikey with one needle per block; the diagnostics below also include TriviaQA. C3 and GD share the live window, FIFO evictions, and per-stream row capacity. C3 uses its jointly trained writer and gated reader; the main GD baseline reads rows through plain softmax. At each eviction, C3 applies its learned forward update, while GD fits the carried state using supervision from only the newest evicted block.

Supervision. Following Cartridges (Eyuboglu et al., 2025), self-study generates requests to summarize, structure, question, or use observed context, then answers with that context available. GD distills the full-history teacher on these request–answer pairs (self-study pairs). Oracle GD instead receives the future test questions and gold answers, isolating acquisition and retention when the future queries are known. Neither C3 nor deployable self-study GD receives this privileged supervision.

Optimization. Each consolidation runs 512 Adam steps at batch size 8 with a fresh optimizer; state parameters persist across consolidations. We minimize $\mathrm{KL}(p_{\mathrm{teacher}} \| p_{\mathrm{student}})$ at temperature 1 over all question and answer tokens, without cross-entropy or early stopping. The teacher sees the full history; the student sees the sinks, live window, and state. We vary supervision volume at fixed optimizer steps, using the first n self-study pairs from the newest evicted block and none from earlier blocks. Rows initialize with keys from a fixed, unrelated passage and zero values. Learning rates are 3×10^{-2} on NIAH and 10^{-2} on TriviaQA, including the oracle, at every supervision volume.

Evaluation and compute. We evaluate each question after every consolidation following its arrival. NIAH measures needle-value recovery; TriviaQA measures gold-alias inclusion within 64 generated tokens, except in the substrate comparison below. Immediate recall measures acquisition, subsequent recall measures retention, and live-window accuracy measures interference. Deployment compute includes self-study generation and all consolidation updates. Intervals are bootstrapped over documents at 95%.

Acquisition and retention. Figure 8 follows each block through the stream. The left panels measure newly evicted blocks immediately after consolidation; the middle panels track the first block through subsequent updates; the right panels report final recall by block age, with age zero denoting the live window.

With abundant self-study, GD acquires new blocks well throughout the stream but rapidly loses earlier information. The first block's recall collapses after the next consolidation on NIAH and approaches the no-memory baseline on TriviaQA. Oracle GD likewise acquires the newest NIAH needle almost perfectly, then loses it at the next update. Supplying the future questions therefore does not resolve forgetting under this protocol: each fit targets the newest block without an objective for preserving earlier predictions. The ungated state also impairs live-window recall.

C3 retains the first block through successive consolidations, while acquisition declines for later blocks. This is consistent with its competitive merge: new content must win rows already held by earlier observations. Its final advantage comes primarily from retention, despite weaker immediate acquisition than GD later in the stream on NIAH and throughout the stream on TriviaQA. These diagnostics identify different bottlenecks: preserving earlier information for GD, and admitting new information into an occupied state for C3.

C.2 WHAT GD LEARNS FROM ITS SUPERVISION

Figure 9 isolates acquisition by evaluating each block immediately after consolidation. GD outperforms C3 on the fitted self-study questions and, with sufficient pairs, on held-out questions from the same generator. C3 was not trained on this distribution and remains near the foreign-document-state control, whose nonzero token F1 reflects shared answer wording without the relevant history.

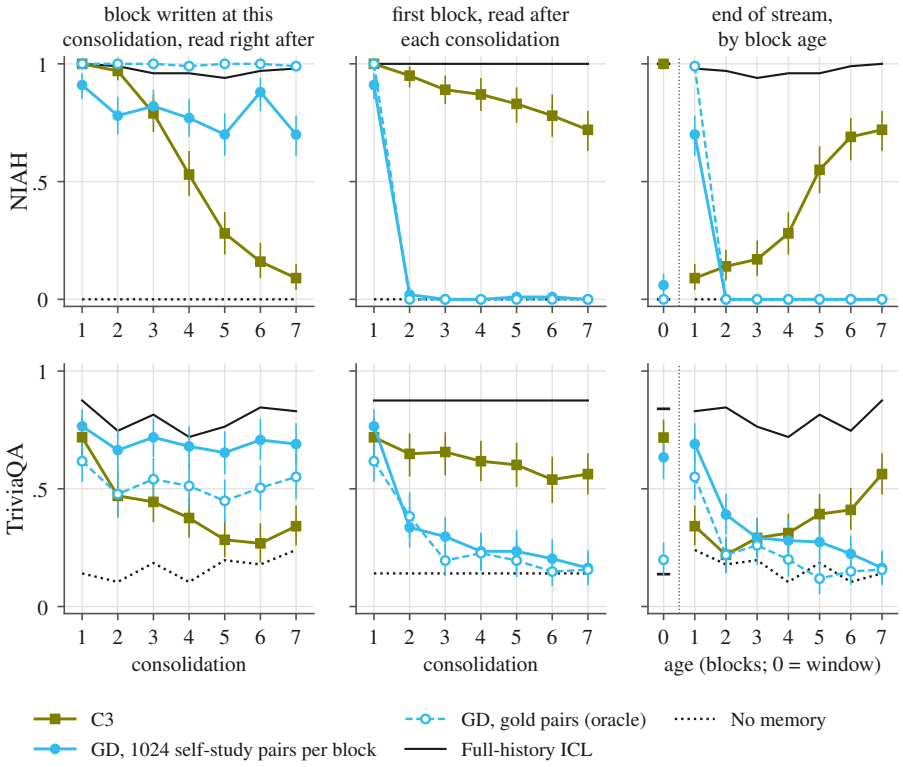

Figure 8: **Acquisition and retention over the stream.** C3, self-study GD (1024 pairs per block), and oracle GD on NIAH (top) and TriviaQA (bottom). Left: immediate recall of each newly evicted block. Middle: recall of the first block after successive consolidations. Right: final recall by block age; age zero is the live window. Solid black lines show full-history ICL; dotted lines show no memory. Each task uses 100 eight-block documents, with 95% bootstrap intervals.

Acquiring information useful for the actual test questions requires substantially more self-study pairs. Thus, fitting and generalizing within the self-study distribution need not yield comparable future-use generalization. More supervision improves acquisition in these experiments, but they do not establish that unlimited self-study would cover the deployment distribution. C3 instead learns from curated future-use supervision offline, avoiding per-block supervision generation at deployment.

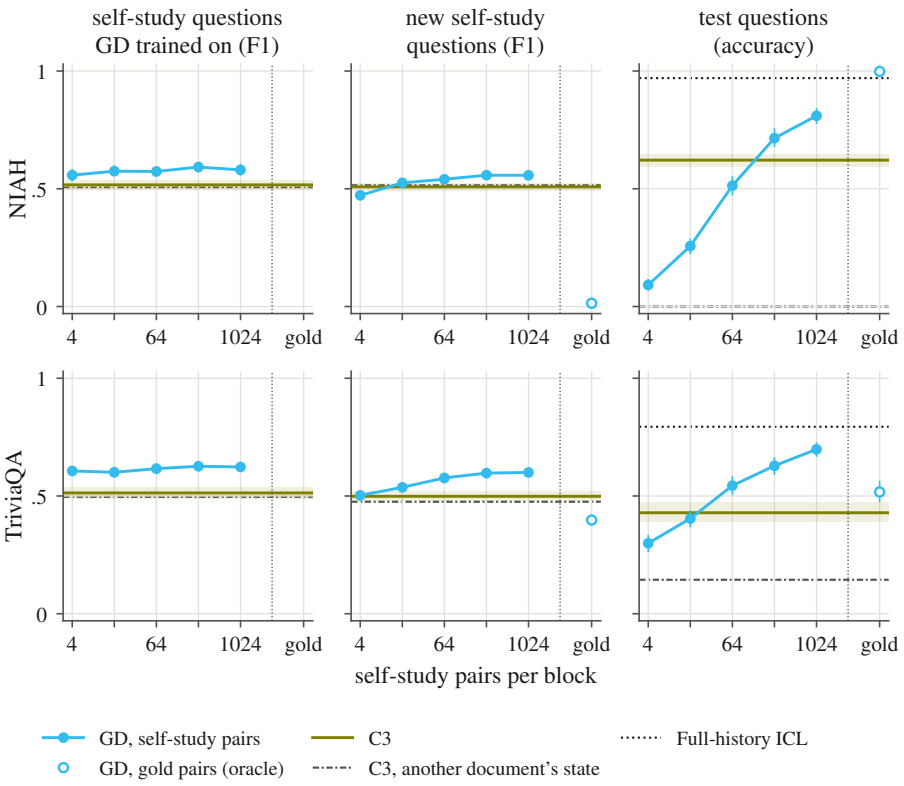

Figure 9: **Acquisition on self-study and future test questions.** NIAH (top) and TriviaQA (bottom), evaluated immediately after consolidation. The horizontal axis varies self-study pairs per block. Left and middle: token F1 against teacher answers on four fitted and four held-out self-study questions per document. Oracle GD sees neither set and is shown once, in the middle. Right: test accuracy, using needle-value recovery or gold-alias inclusion within 64 generated tokens. Dash-dotted lines use C3 with a foreign document's state; dotted lines show full-history ICL. Each task uses 100 eight-block documents, with 95% bootstrap intervals.

Oracle acquisition on TriviaQA. Oracle GD nearly solves immediate NIAH recall but remains below full-history ICL on TriviaQA despite low KL on the fitted sequences. A possible explanation is a mismatch between teacher-forced fitting and free generation. Training pairs contain bare gold answers; the TriviaQA teacher often generates a sentence prefix first, taking decoding outside the fitted prefixes. Oracle accuracy is higher when the teacher answers directly, although gold aliases and the teacher's wording can still differ.

The oracle also repeatedly fits very few pairs at the self-study learning rate. Its state harms self-study answers and live-window recall, and a foreign document's oracle state can perform below no memory. These observations suggest interference as another possible contributor, but do not isolate a cause. Teacher-generated answer targets, answer-only loss, and a separately tuned oracle learning rate remain untested controls.

C.3 Memory substrates and learned readers

Does C3's advantage depend on the state representation, the trained reader, or the learned update? Figure 10 compares forward writers, GD, and GD with a trained reader across four memory substrates at a matched state budget.

State budget. Qwen3-1.7B has $L = 28$ layers and $H = 8$ KV heads of dimension $d_h = 128$. Each substrate holds approximately 1024 KV-token equivalents, or 5.9×10^7 values per document, with $2LHd_h$ values per equivalent. We distinguish the document-specific *state* from the *reader parameters*, which are shared across documents and fixed at deployment. Below, q is a normalized query before rotary embedding, h is an MLP input, and layer indices are omitted.

Attention rows. C3 stores $R = 1024$ key–value rows per layer and KV head, shared by the corresponding query heads. A separate softmax reads these rows and adds their contribution to ordinary attention, without taking attention mass from the live window. C3 uses the trained, gated reader of Section 3.2. Generic GD instead uses

$$r(q) = \sum_r \mathrm{softmax}_r\big(q^\top k_r/\sqrt{d_h}\big)\, v_r,$$

without learned gating or scaling.

Prefix tokens. The Cartridges parameterization (Eyuboglu et al., 2025) places the same number of free KV pairs before the retained context. The model's own attention reads the prefix $\mathcal{P}$ and retained context $\mathcal{C}$ jointly,

$$o(q) = \sum_{j \in \mathcal{P} \cup \mathcal{C}} \mathrm{softmax}_j\big(q^\top k_j/\sqrt{d_h}\big)\, v_j,$$

with rotary embeddings omitted. Unlike attention rows, the prefix competes with the live window for attention mass and requires no additional reader parameters.

MLP rows. This substrate moves the row read from attention to the MLP. Keys and values are stacked across heads, $k_r, v_r \in \mathbb{R}^{Hd_h}$, and the MLP input forms a query $u = N(\widetilde{W}_Q h)$ with one query head per KV head, using the model's per-head query normalization. The read is

$$\rho(h) = \sum_r \mathrm{softmax}_r\Big(\beta\, \frac{u^\top P_K k_r}{\sqrt{Hd_h}}\Big)\, P_V v_r.$$

A per-head gate $\sigma(w_i^\top u_i + b)$ gives $\tilde{\rho}(h)$, and the MLP output becomes $\mathrm{MLP}(h) + \widetilde{W}_O \tilde{\rho}(h)$. Reader parameters comprise the projections, P_K, P_V, β, and the gate. Before training, P_K and P_V are identities, the projections are copied from attention, and the gate is the constant $\sigma(-2)$.

Gram core. A matrix $G \in \mathbb{R}^{d_c \times d_c}$ per layer, with $d_c = 1448$, parameterizes updates to all three MLP projections:

$$W_p x \;\to\; W_p x + \gamma D_p G D_p^{A\top} x, \qquad p \in \{\mathrm{gate}, \mathrm{up}, \mathrm{down}\}.$$

The decoders have dimensions $D_p \in \mathbb{R}^{d_p^{\mathrm{out}} \times d_c}$ and $D_p^A \in \mathbb{R}^{d_p^{\mathrm{in}} \times d_c}$. The core is shared across projections within a layer: it is the trainable middle factor of a factorized weight update whose outer factors remain fixed at deployment. Before training, decoders are random and $\gamma = 0.1$. This ungated update affects every token, including the live window and query.

Forward writers. We train each writer jointly with its reader using the C3 objective: 2000 steps from scratch followed by a 500-step decay. The attention-row writer is C3's; the MLP-row writer uses the same write and competitive merge through the MLP read. The Gram-core writer instead forms gated outer products and merges blocks by their accumulated mass:

$$\tilde{G} = \frac{\sum_j g_j c_j a_j^\top}{\tilde{m}}, \qquad G' = \frac{mG + \tilde{m}\tilde{G}}{m + \tilde{m}}, \qquad \tilde{m} = \sum_j g_j.$$

The gate g_j and codes c_j, a_j are computed from the evicted token's key, value, and position; m is the carried mass. The training recipe was developed for attention rows and reused for the other substrates. We did not train a prefix-token writer.

GD and trained-reader controls. GD uses 1024 self-study pairs per block and the optimization protocol above. Generic GD uses each substrate's reader before offline training. Rows initialize as above, prefix keys and values come from the fixed unrelated passage, and Gram cores start at zero. Row learning rates match the main protocol; prefix tokens use 3×10^{-2} and Gram cores use 10^{-2}.

The trained-reader control substitutes reader parameters learned jointly with a forward writer, taken before its decay stage. GD still starts from the same initial state as generic GD, rather than a forward-written state. Learning rates are unchanged except for the Gram core, which uses 10^{-1} and diverges; this cell is omitted from the figure.

Metrics. Panels report final accuracy on evicted blocks, immediate acquisition, recall after the next consolidation, and live-window accuracy. The attention-row NIAH cells match C3 and the highest-supervision GD cells in Figure 2. TriviaQA uses a 24-token generation limit here, so its values are not directly comparable with the 64-token diagnostics above.

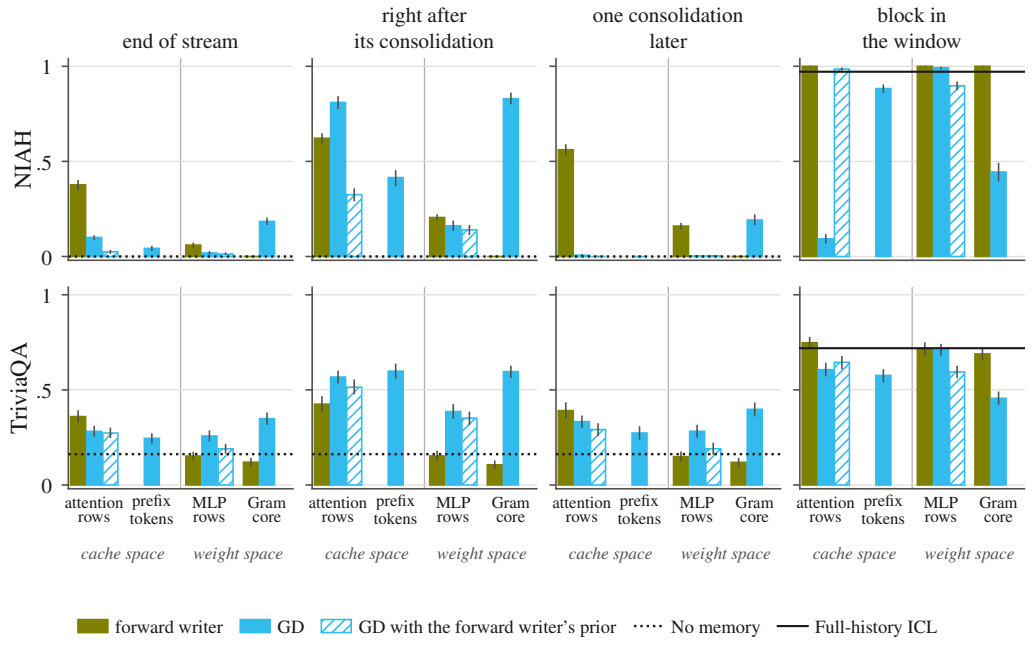

Figure 10: **Memory substrates and learners.** NIAH (top) and TriviaQA (bottom), with approximately 1024 KV-token equivalents per state. Bars compare forward writers (olive), generic GD (blue), and GD with a trained reader (hatched). GD uses 1024 self-study pairs per block. Panels show final evicted-block accuracy, immediate recall, recall after the next update, and live-window accuracy. Dotted lines show no memory; solid lines show full-history ICL. Missing bars indicate untrained variants or the diverged Gram-core control. TriviaQA uses 24-token decoding here, versus 64 above. Each task uses 100 eight-block documents, with 95% bootstrap intervals.

The learner and substrate interact. The forward writer performs best on attention rows and retains little evicted information in either MLP rows or the Gram core, although both preserve the live window. GD instead achieves its strongest final performance on the Gram core. On NIAH, this is the only tested substrate where generic GD retains appreciable information after the next update, but it also causes the greatest live-window interference. These results do not necessarily establish a limit of forward writing into weights: the training recipe was developed for attention rows, and GD demonstrates that the Gram core can store useful information. We hope to further characterize the interplay between substrate and learner in future work.

The forward writer's prior does not rescue GD. We test whether GD improves when given the reader parameters learned during forward-writer training. Across both row substrates, GD with this prior finishes at or below generic GD. For attention rows, the trained gate reduces live-window interference but weakens acquisition and does not prevent forgetting; the trained MLP reader likewise provides no improvement. These results show that inheriting an offline-learned reader alone does not resolve GD's continual-learning failures. This control does not initialize from a forward-written state or train the reader specifically for GD.

Sharing the attention softmax does not resolve GD's forgetting. We also test whether GD's retention failure comes from reading its state through separate attention. Prefix-token GD instead

places learned KV tokens in the same attention softmax as the live context, following the Cartridges parameterization (Eyuboglu et al., 2025). It finishes below attention-row GD on both tasks: acquisition is comparable on TriviaQA and weaker on NIAH, and most acquired information is lost after the next update. Shared attention reduces NIAH live-window interference, but does not resolve forgetting.

The TriviaQA tradeoff. GD on the Gram core matches C3's final TriviaQA accuracy and recall after the next update. Its foreign-document state remains below no memory, indicating that the gain depends on the document. This retention comes with substantial live-window interference and the deployment cost of repeated optimization. C3 preserves the live window at far lower deployment cost, while leading all tested GD configurations on final NIAH accuracy.

D Architecture ablations

We first isolate the core choices in C3: normalized writing, competitive merging, gated reading, and context distillation. Additional ablations examine variants and deployment settings.

D.1 Core ablations

Protocol. We train Qwen3-1.7B with $R = 1024$ rows per layer on 16.5K-token streams of eight $B = 2048$ blocks, using FIFO eviction with three sink tokens. Training uses 1,000 optimizer steps, learning rate 10^{-3}, a warmup–stable–decay schedule, and eight streams per step. The data are ProLong document streams with 20% document QA, as in Section 3, without the final task-mixture decay stage. Variants change one component relative to their reference: the default configuration for most comparisons, and the averaging merge for the sum-write variants. The latter do not define the normalized writer's absorbed mass used by competitive merging. Each variant uses one seed; we treat closure differences below 0.03 as inconclusive.

Evaluation metrics. We measure recovery of evicted information with *LongPPL* (**?**), which focuses on predictions that benefit from long context. For key-token loss $\mathcal{L}$, we define

$$\text{Closure} = \frac{\mathcal{L}_{\text{evict}} - \mathcal{L}_{\text{state}}}{\mathcal{L}_{\text{evict}} - \mathcal{L}_{\text{full}}}.$$

We evaluate on held-out ProLong-QA, SQuAD, TriviaQA, and needle-in-a-haystack (NIAH) streams (Hsieh et al., 2024). A *foreign-state* control uses memory from an unrelated stream to test whether recovery depends on the consolidated history. We measure *non-interference* through changes in in-window retrieval and GSM8K accuracy, probing preservation of ICL and parametric capabilities, respectively. We also report greedy TriviaQA answer accuracy after the seventh block.

Table 1 summarizes the core design choices; Table 2 reports all variants.

Table 1: **Ablations of the C3 functional form.** Closure measures recovery of evicted-context LongPPL loss; non-interference reports the change in task accuracy when the state is attached, with 0 indicating no damage.

	Closure			Non-interference (Δ accuracy)	
	ProLong-QA ↑	foreign ≈ 0	NIAH ↑	in-window NIAH	GSM8K
C3	0.727	0.029	0.373	0.00	−0.01
averaging merge	0.633	0.019	0.298	−0.06	−0.01
+ sum write	0.285	0.180	0.077	−0.24	−0.06
linear read	0.292	0.149	0.096	−0.24	0.00
no read gate	0.706	−0.010	0.364	−0.34	−0.16
unweighted objective	0.597	0.000	0.281	+0.02	+0.04

D.1.1 Writer

Replacing token softmax with a linear-attention write (Katharopoulos et al., 2020) gives

$$\tilde{v}_r = \sum_j g_j \, (\beta_r^\top k_j) \, v_j, \qquad \tilde{k}_r = \frac{\sum_j g_j \, (\beta_r^\top k_j) \, k_j}{\sum_j g_j \, (\beta_r^\top k_j)}.$$

Values become signed, unnormalized sums whose scale depends on the block. Relative to the averaging-merge reference, this reduces closure across all evaluated streams and increases foreign-state recovery, indicating weaker dependence on the consolidated history.

D.1.2 MERGE

Weighting old and incoming rows by the total write mass they represent gives

$$(k'_r, v'_r) = \frac{n_r\,(k_r, v_r) + \tilde{n}_r\,(\tilde{k}_r, \tilde{v}_r)}{n_r + \tilde{n}_r}.$$

Each row becomes a weighted average of successive candidates. This reduces closure on all streams and decoded TriviaQA accuracy, with a modest increase in in-window interference.

D.1.3 READER

Replacing row softmax with a kernel feature map gives

$$r(q) = \gamma(q)\,\frac{\phi(q)^\top \sum_r \phi(k'_r)\,v'_r}{\phi(q)^\top \sum_r \phi(k'_r)}, \qquad \phi = \mathrm{elu} + 1.$$

This preserves non-negative weights and a positive denominator, but yields higher distillation loss and substantially lower closure. Signed features with normalization diverge as the denominator crosses zero. Fixing $\gamma(q) \equiv 1$, while retaining the per-layer gain, largely preserves closure but sharply increases damage to in-window retrieval and GSM8K. The gate primarily controls interference with existing capabilities.

D.1.4 OBJECTIVE

On text continuations, we weight each position by the divergence between the frozen teacher's full-history and retained-context predictions,

$$w_{t,i} = \mathrm{KL}(p_\theta(\cdot \mid h_t, x_{<i}) \,\|\, p_\theta(\cdot \mid c_t, x_{<i}))\,.$$

This emphasizes positions whose predictions depend on evicted information. Uniform weights reduce closure across streams without increasing interference.

D.2 ADDITIONAL ABLATIONS

Table 2 includes the core comparisons above and the additional variants below.

D.2.1 WRITER

Fixed recency decay. Multiplying each token's weight by λ^{T-1-j} for block length T and $\lambda = 0.999$ leaves ProLong-QA and NIAH closure essentially unchanged and increases in-window interference. Fixed decay therefore does not recover the normalized writer's performance.

Learned per-token decay. A zero-initialized head predicts retention $\alpha_j = \sigma\big(\mathrm{MLP}(d_j) + \mathrm{logit}\,\lambda\big)$, weighting token j by $g_j \prod_{i>j} \alpha_i$. The head starts at the fixed decay and learns how later tokens attenuate earlier ones; in the softmax writer, decay affects both assignment logits and absorbed mass. Learned decay does not recover the sum writer's low closure, suggesting that recency weighting alone does not resolve its failure. With the normalized writer, closure remains within the inconclusive band, while decoded TriviaQA accuracy declines and in-window interference increases. We therefore retain the simpler writer without learned decay.

Other writer variants. Doubling assignment temperature to $\tau = 0.1$ gives mixed closure changes and lower decoded TriviaQA accuracy. Replacing pooled content keys with block-derived span keys reduces closure throughout, supporting addresses based on the content stored in each row. Re-querying evicted tokens for a refinement pass, selecting 32-token chunks before individual tokens, and using residual values yield no consistent closure improvement. Experiments also found that copying individual tokens into rows performed worse. We did not evaluate the delta-rule write (Yang et al., 2025c) at $R = 1024$ because the available kernel limits the key dimension to 256.

Table 2: **All functional-form ablations** (Qwen3-1.7B, $R = 1024$, 16.5K streams, 1,000 steps, one seed). Closure on LongPPL key tokens (ProLong-QA, SQuAD, TriviaQA, NIAH streams; 0 = retained window without a state, 1 = full context); foreign = ProLong-QA closure with a state from an unrelated stream; decoded TriviaQA = greedy answer accuracy after the seventh block; non-interference = signed change in accuracy with the state attached (in-window NIAH at age 0; GSM8K, $n = 100$). The three sum-write cells run under the averaging merge, so their reference row is *averaging merge*. Differences below 0.03 closure are within run-to-run variation.

	Closure					Decoded	Non-interference (Δ)	
	ProLong-QA	foreign	SQuAD	TriviaQA	NIAH	TriviaQA	in-window NIAH	GSM8K
C3 (recipe)	0.727	0.029	0.630	0.650	0.373	0.435	0.00	−0.01
Writer								
sum write (linear attention; averaging merge)	0.285	0.180	0.197	0.520	0.077	0.253	−0.24	−0.06
sum write + fixed recency decay $\lambda = 0.999$ (averaging merge)	0.285	0.181	0.182	0.564	0.073	0.263	−0.44	0.00
sum write + learned per-token decay (init $\lambda = 0.999$; averaging merge)	0.281	0.186	0.212	0.547	0.059	0.273	−0.34	+0.02
softmax write + learned per-token decay (init $\lambda = 0.9999$)	0.730	0.028	0.617	0.626	0.395	0.396	−0.06	+0.08
assignment temperature $\tau = 0.1$ (recipe 0.05)	0.710	0.027	0.586	0.698	0.362	0.380	+0.02	+0.02
span row keys	0.452	0.076	0.342	0.513	0.153	0.378	−0.22	−0.05
one refinement pass	0.704	0.041	0.643	0.668	0.347	0.396	−0.12	+0.02
chunk-then-token selection (32)	0.743	0.031	0.618	0.617	0.387	0.407	−0.04	+0.07
residual values	0.726	0.037	0.621	0.657	0.384	0.441	−0.10	+0.03
Consolidation rule								
averaging merge	0.633	0.019	0.418	0.600	0.298	0.314	−0.06	−0.01
competitive merge, incumbent mass decayed ($\lambda = 0.9$)	0.702	0.029	0.548	0.665	0.380	0.380	−0.04	−0.01
learned recurrent merge (joint softmax)	0.655	0.040	0.449	0.562	0.318	0.343	−0.06	+0.03
learned soft gate, soft-contest init	0.731	0.030	0.563	0.687	0.419	0.349	−0.04	+0.10
learned soft gate, average init	0.698	0.038	−3.511	0.665	0.365	0.353	−0.04	0.00
Reader								
linear read, kernelized ($\phi = \mathrm{elu} + 1$)	0.292	0.149	0.195	0.503	0.096	0.263	−0.24	0.00
learned null row	0.722	0.027	0.640	0.630	0.383	0.421	+0.02	0.00
no read gate ($\gamma \equiv 1$)	0.706	−0.010	0.583	0.648	0.364	0.409	−0.34	−0.16
neutral read-gain init (10^{-2})	0.676	0.052	0.514	0.541	0.325	0.349	−0.16	+0.02
Objective								
unweighted objective	0.597	0.000	0.358	0.510	0.281	0.372	+0.02	+0.04
KL + cross-entropy	0.749	0.115	0.700	0.742	0.461	0.402	−0.02	+0.05
State size								
$R = 512$	0.659	0.043	0.515	0.610	0.269	0.370	−0.04	+0.03
$R = 2048$	0.803	0.013	0.692	0.629	0.495	0.458	−0.06	+0.06
$R = 4096$	0.863	0.010	0.743	0.632	0.647	0.472	0.00	+0.04
$R = 8192$	0.895	0.004	0.792	0.681	0.713	0.538	−0.08	+0.06
Block schedule								
block $B = 1024$	0.758	0.006	0.599	0.728	0.409	0.468	−0.14	+0.06
block $B = 4096$	0.671	0.035	0.533	0.623	0.340	0.320	−0.02	+0.10
consolidate every second block	0.706	0.028	0.617	0.632	0.373	0.382	−0.04	+0.01

D.2.2 MERGE

Incumbent decay. Decaying the incumbent's mass before selection, $\tilde{g}_r > \lambda g_r$ with $\lambda = 0.9$, gives no consistent closure improvement and lowers decoded TriviaQA accuracy. We retain the undecayed competitive merge.

Learned recurrent merge. We include carried rows in the writer's softmax, scored by their keys and gated by their mass plus a learned bias. Each output row combines incoming tokens and previous rows, replacing the separate merge step. Performance is comparable to averaging, and updates must be applied sequentially across blocks.

Learned soft gate. A learned gate blends old and incoming rows,

$$(k'_r, v'_r, g'_r) = (1 - u_r)\,(k_r, v_r, g_r) + u_r\,(\tilde{k}_r, \tilde{v}_r, \tilde{g}_r), \qquad u_r = \sigma\big(w^\top \phi_r + b\big).$$

The per-layer parameters w, b act on features

$$\phi_r = \big[(\tilde{g}_r - g_r)/\tau_m,\ \cos(\tilde{k}_r, k_r),\ \log g_r,\ \log \tilde{g}_r,\ \log(1 + \mathrm{use}_r)\big],$$

where use_r counts reads addressing the row. Initializing $w = e_1$ and $b = 0$ recovers the sigmoid relaxation of competitive selection. This gives mixed closure changes and lower decoded TriviaQA accuracy, without retaining associative composition. Initializing instead at equal mixing ($w = 0$, $b = 0$) produces late-training instability on SQuAD, including with foreign state, while other stream evaluations remain comparable.

D.2.3 READER

Null row. Adding a learned key–value pair (k_0, v_0) to receive attention when no stored row matches yields no meaningful closure improvement, consistent with the query gate already allowing the model to suppress memory reads.

Gain initialization. Initializing the per-layer read gain at 10^{-2} rather than 1 reduces closure across streams and decoded TriviaQA accuracy at the same training budget.

D.2.4 OBJECTIVE

Cross-entropy. Adding next-token cross-entropy, $\mathcal{L} = \mathcal{L}_{\mathrm{C3}} + \lambda_{\mathrm{CE}}\mathcal{L}_{\mathrm{CE}}$ with $\lambda_{\mathrm{CE}} = 1$, improves closure at 1.7B but increases foreign-state recovery and lowers decoded TriviaQA accuracy. At 8B, it also lowers decoded NIAH accuracy relative to both reference seeds. We therefore retain pure distillation, which directly targets preservation of the predictions enabled by the full context.

D.2.5 STATE SIZE AND BLOCK SCHEDULE

State size. Increasing capacity from $R = 512$ to 8192 monotonically improves ProLong-QA, SQuAD, and NIAH closure and decoded TriviaQA accuracy; TriviaQA closure changes comparatively little. Section 4 reports state scaling at 8B.

Block size. Halving the default block size to $B = 1024$ improves closure on most streams and decoded TriviaQA accuracy, but increases in-window interference. Doubling it to 4096 reduces closure throughout. We retain $B = 2048$ to balance recovery and interference. We did not evaluate $B = 512$ because it cannot accommodate the stream prefix and a needle together.

Consolidation cadence. Combining consecutive blocks in each update, $m_{t+1} = C_\phi(m_t, b_{t-1} \oplus b_t)$, leaves closure largely unchanged but lowers decoded TriviaQA accuracy.

E ADDITIONAL RESULTS

E.1 EVALUATION COVERAGE AND PER-TASK COMPARISONS

Figure 11 extends the main matched-budget comparison on Qwen3-8B to a broader set of long-range modelling tasks. Figure 12 separately compares Knowledge Seeding with C3 at 1.7B. We use 8K-token streams to keep deployment-time optimization tractable to evaluate. Figure 7 reports consolidation time per stream, measured on a single H100 for Knowledge Seeding on Qwen3-8B at 8K tokens, alongside extrapolated GPU-hours per stream for each task. Figure 13 compares C3 with the released one-shot SHINE checkpoint at matched context-dependent storage, with the budget set by SHINE's released weights. These comparisons keep the continual and one-shot evaluation settings explicit.

E.2 CONTINUAL ACQUISITION AND RETENTION

Figure 14 separates accuracy by the age of the evidence. This distinguishes the acquisition of newly evicted content from the retention of information already stored, which end-of-stream averages combine. C3 favors older evidence rather than exhibiting recency bias.

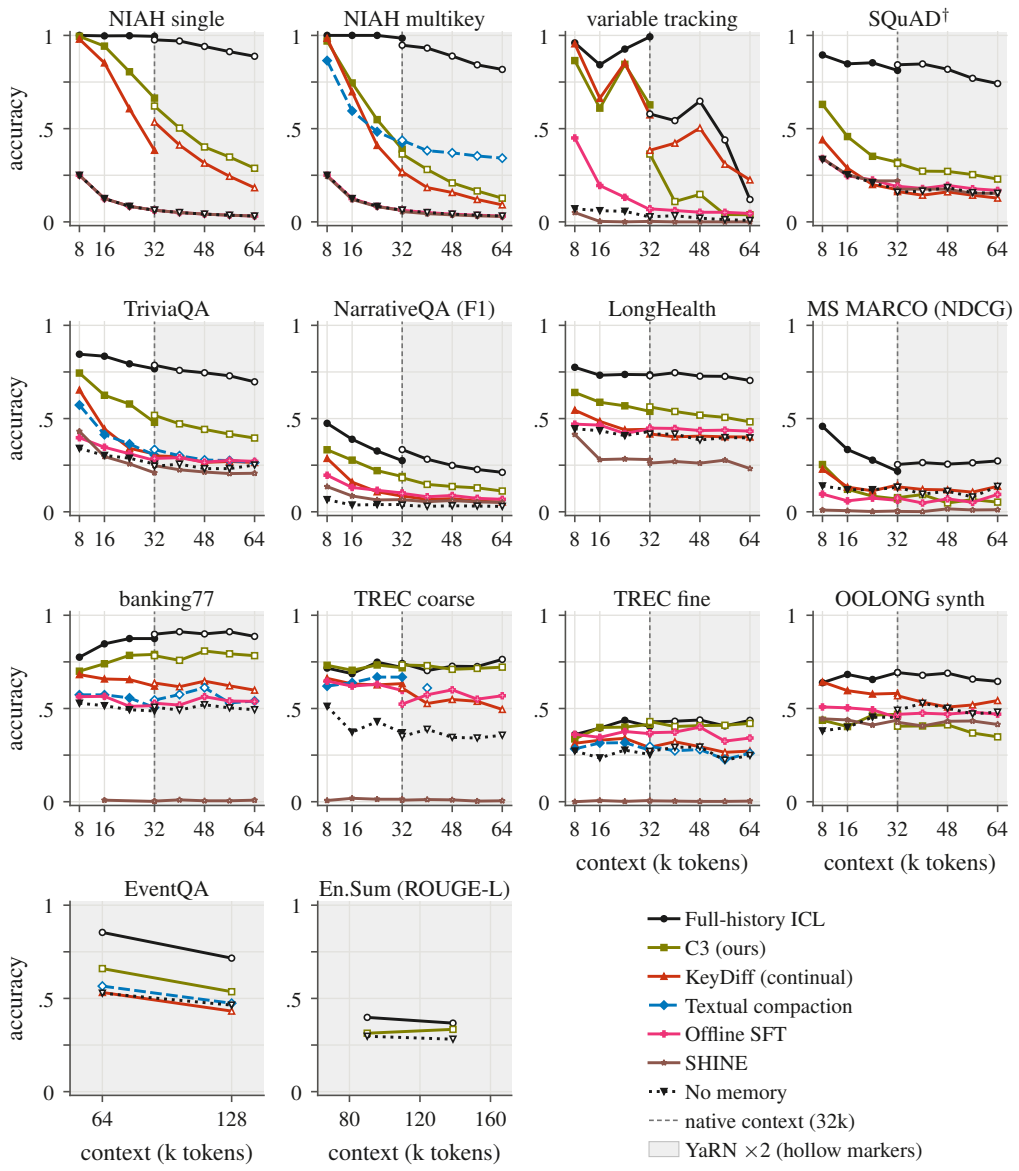

Figure 11: **Expanded Qwen3-8B comparisons at $r = 2048$.**

E.3 NON-INTERFERENCE

Figure 15 extends the analysis of interference with base-model capabilities on GSM8K to SHINE and offline SFT. Both show substantially greater interference than C3.

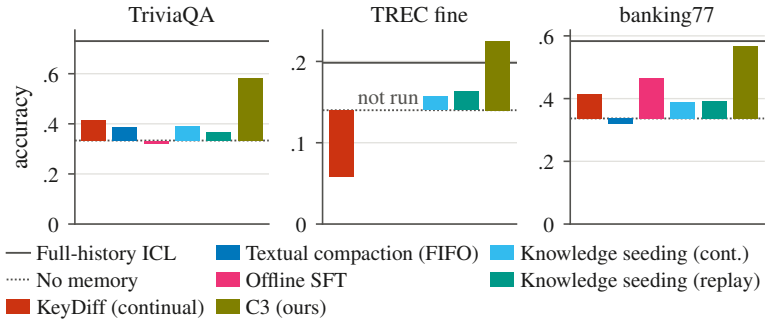

Figure 12: **Knowledge Seeding and C3 on Qwen3-1.7B.** Accuracy on four-block, 8K-token streams of TriviaQA, TREC fine, and banking77, pooled over 50 streams. Lines show full-history ICL and no memory. Methods use comparable state budgets. We evaluate Knowledge Seeding at 1.7B, where we can afford more optimization steps to ensure convergence.

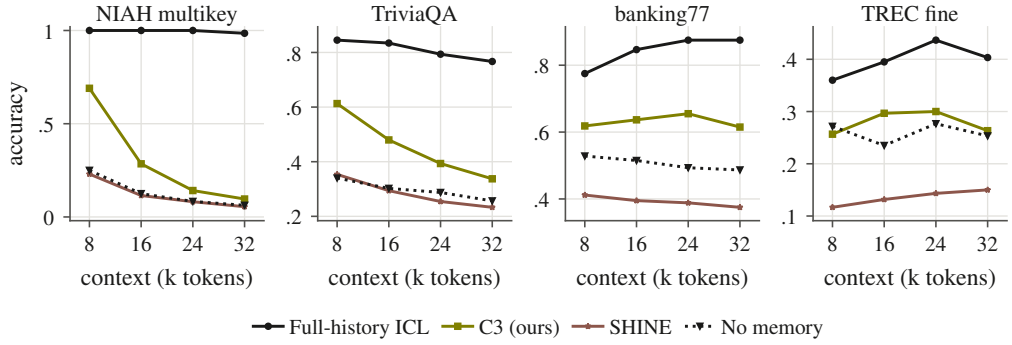

Figure 13: **Comparison of C3 with SHINE** (Liu et al., 2026). We compare C3 and SHINE accuracy as a function of context length at matched context-dependent storage, with the budget set by SHINE's released weights. We evaluate the released SHINE checkpoint in its one-shot setting: all context except the retained 2048-token window is compressed at once. C3 updates its matched-size state continually.

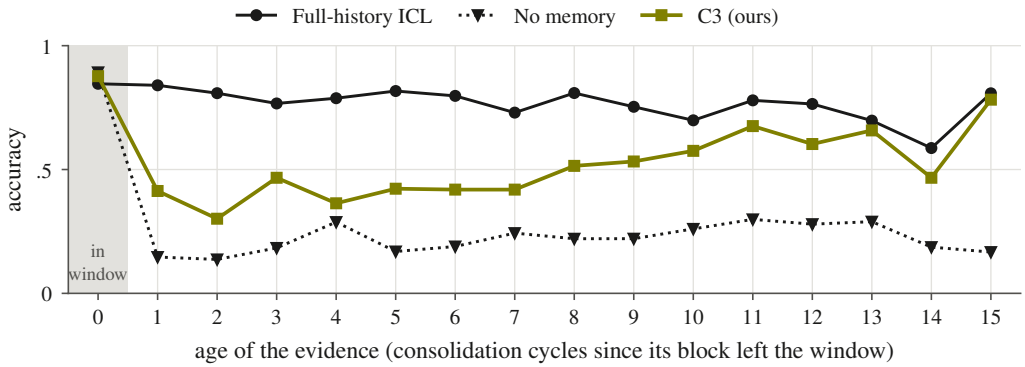

Figure 14: **TriviaQA accuracy by evidence age on Qwen3-8B at 32K tokens.** Each stream contains 16 blocks of 2048 tokens; evidence age is the number of consolidation cycles between the document and the question.

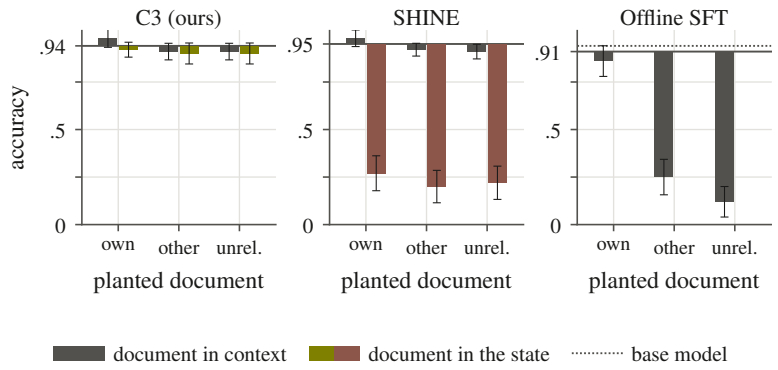

Figure 15: **Interference with base-model GSM8K performance across context consolidation algorithms.** On Qwen3-8B, the planted document is the item's own reference solution (*own*), another item's reference solution (*other*), or an unrelated essay block (*unrel.*). Grey bars show results with the document's block in the context window; coloured bars show results after the same block is consolidated into the system's memory before the question. Each bar starts at the system's accuracy without a document (the horizontal line) and ends at its accuracy with the document.