SOME MEMORIES MUST FADE: PERSISTENCE AND DECAY OF USER MEMORIES IN MODEL PARAMETERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Parametric memory allows Large Language Models (LLMs) to internalize knowledge directly into their weights, offering a potentially powerful alternative to retrieval-based memory (RAG). However, existing approaches to parametric memory are difficult to deploy as they largely mirror the Continual Learning (CL) objective of retaining information indefinitely. This is neither sustainable nor sufficient for a memory system, which must govern knowledge across its lifecycle by retaining stable facts, reinforcing recurring observations, updating changed beliefs, and allowing transient information to fade. We propose a system for parametric memory management consisting of a memory substrate (handling information storage and forgetting) and a learned policy (what to remember, how deeply, and for how long). Our substrate is a fixed-capacity LoRA adapter that stores memories in sparse k-hot combinations of adapter ranks to limit interference. This allows each rank and its stored information to maintain an independently controlled temporal decay, enabling outdated information to fade while long-term facts persist. On this substrate, we train a lightweight 2M-parameter controller that governs each incoming user event to ensure the memory system reflects the user's true, updated belief state. To evaluate how memory systems manage user memory, we introduce a controlled dataset of long-horizon user trajectories with varying persistence of facts based on changing belief states and user-behavioral shifts, with probe queries that test recall while information is valid and abstention after it expires. Our policy learns to effectively manage user memory over its entire lifecycle, balancing recall and abstention to achieve 86% of oracle performance, unlike CL and RAG baselines that primarily default to one of the two behaviors. By providing the first system to manage memory directly in model weights, this work establishes a foundation for long-horizon parametric memory.

1 INTRODUCTION

Large Language Models (LLMs) increasingly operate over long horizons as agents and personalized assistants, necessitating memory systems to match evolving user preferences (Ding et al., 2026; Liu et al., 2025). Currently, retrieval-based memory is widely adopted for its simplicity—storing information externally and injecting it into the context when relevant (Lewis et al., 2021; Kang et al., 2025). Parametric memory instead encodes information directly into the model's weights, allowing LLMs to leverage memory implicitly without cluttering the prompt context (Wang et al., 2025b; Behrouz et al., 2024; Su et al., 2025). However, this approach remains difficult to deploy in practice due to its rigidity in adapting to the evolving nature of user memory. Current approaches largely inherit the objective of continual learning (CL), focusing on how to acquire new knowledge without forgetting (McCloskey & Cohen, 1989; Kirkpatrick et al., 2017). We demonstrate that this objective is insufficient for long-horizon assistants. Memory management instead asks which knowledge should be retained, reinforced, overwritten, or forgotten, taking downstream use into account. For instance, *I had a bagel for breakfast* should not remain valid even the next day, while *I work as a teacher* should likely be relevant information for years. External stores expose lifecycle operations directly through addition, update, and deletion, but model parameters do not (Yan et al., 2026; Yu et al., 2026). Consequently, in our evaluation, no CL method is able to represent outdated information even though they store values well (0.82 retrieval with PCGrad (Yu et al., 2020)).

We hypothesize that usable parametric memory requires separating the storage dynamics (how information is stored, and isolated in the substrate) from its policy (what to store, how deeply, and for how long). For the substrate dynamics, we design a fixed-size LoRA substrate (Hu et al., 2021) where

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.

each fact is routed to a k-hot combination of adapter ranks (Zou et al., 2025). This bounds gradient interference between unrelated facts, while an independent temporal decay rate on each rank allows stale information to fade (Ebbinghaus, 1885) without explicit unlearning. For the policy, we train a lightweight 2M-parameter controller that decides, per incoming event, whether to write, at what gradient depth, and with what lifespan. Stable information receives deep durable writes, repeated routines see accumulating writes, while transient events see short or no lifespans.

Evaluating these lifecycle decisions requires ground truth that existing memory benchmarks do not provide at the event level. We therefore construct 909 controlled, long-horizon user trajectories with lifecycle annotations capturing validity, persistence, reinforcement, and contradiction, along with programmatically injected queries that probe per-event management decisions. On this testbed, our learned management system reaches 0.56 class-balanced accuracy, compared with 0.42 for the strongest CL baseline (PCGrad) and 0.50 for a compute-matched retrieval-based memory manager, recovering 86% of oracle performance. We study two questions: (1) Can a fixed-capacity parametric substrate support reliable information persistence and decay with varying per-fact dynamics? (2) Given such a substrate, can a policy infer from online observations how each fact should be managed? Our contributions are:

- **Task and controlled benchmark** (§2): We formalize parametric memory management around four requirements—valid retention, selective expiration, reinforcement and contradiction, and user-specific dynamics—and construct a dataset of long-horizon user trajectories with event-level lifecycle supervision and diagnostic probes.
- **Memory substrate** (§3): We propose a fixed-capacity LoRA substrate combining sparse k-hot rank allocation with per-rank temporal decay, providing controllable mechanisms for selective persistence, reinforcement, overwrite, and calibrated expiration.
- **Learned management** (§4): We define a $(\text{binary gate}, \text{depth}, \text{lifespan})$ action space over the substrate and train a 2M-parameter controller that recovers 86-92% of oracle performance, demonstrating that effective lifecycle decisions can be inferred from the available online signals.

2 Task: Parametric Memory Management

2.1 Memory Management Requirements

We study parametric memory management for lifelong user-facing agents, where rather than preserving all written information, the system must manage each fact's lifecycle as user knowledge evolves. We identify four requirements:

1. **Valid retention:** Recall stored information while it remains valid.
2. **Selective expiration:** Once information is no longer valid, abstain.
3. **Reinforcement and contradiction:** Repeated observations should strengthen an existing belief, while contradictory observations should weaken or overwrite it.
4. **User-specific dynamics:** Persistence and decay must be inferred from individual history, as the same type of information can be long-term for one user but transient for another.

Success requires controlling when and how individual memories persist, accumulate, change, or expire. Existing long-term memory benchmarks evaluate several related capabilities, including recall and belief revision, but none evaluate these lifecycle requirements at the event level (Table 1).

2.2 Controlled Long-Horizon Trajectories

Evaluating these requirements requires ground truth for each fact at different stages in its lifecycle. Real long-term user histories generally do not provide this supervision. Privacy constraints limit collection, management events such as contradictions and temporary shifts are sparse, and the true validity of individual memories is generally unknown even when long interaction histories are available (Uddin et al., 2026).

We therefore generate a benchmark of 909 controlled, multi-year user trajectories (spanning 700–1,500 days) with dense event-level supervision from Gemini 3.7 Flash. Each trajectory is constructed via a five-stage pipeline: (1) sampling a behavioral motif (e.g., frequent routine changes) and a persona, (2) generating a multi-year backstory, (3) converting the backstory into a chronological event stream, (4) masking entities and injecting ground-truth labels for lifespan and stability, and (5) validating chronological consistency.

	Belief Δ	Expiry	Reinforcement	User-dynamics	Lifecycle GT
ForgetBench (Gu et al., 2026)	◐	✗	✗	✗	✗
LoCoMo (Maharana et al., 2024)	◐	✗	✗	✗	✗
PersonaMem (Jiang et al., 2025)	✓	✗	✗	◐	◐
MemoryAgentBench (Hu et al., 2026)	✓	✗	✗	✗	✗
LongMemEval (Wu et al., 2025)	✓	✗	✗	✗	◐
Memora (Uddin et al., 2026)	✓	✗	◐	◐	✓
Ours	✓	✓	✓	✓	✓

Table 1: Coverage of memory-management properties (✓ full, ◐ partial, ✗ absent). Ours evaluates belief revision, time-based expiry, reinforcement, and user-conditioned lifespans while providing dense lifecycle supervision. Further details are in Appendix C.4.

To prevent the model from relying on pre-trained world knowledge, we replace concrete entities with typed placeholders inside fixed templates (e.g., replacing "apple" with `FRUIT_4` in the template *My favorite fruit is [MASK]*). Both writes and queries use this templated form as input and the model must output the valid value that should be in place of [MASK] or abstain if expired. By stripping away real-world priors, we force the controller to learn generalizable patterns that are not entity-specific or data generation artifacts (not memorizing typical lifespans for entities). The event-level annotations support both evaluation and training. They provide dense supervision over memory validity, lifespans, reinforcement, and contradiction, enabling supervised or imitation learning, dense reward construction, and end-to-end reinforcement learning for memory management (Appendix C).[1]

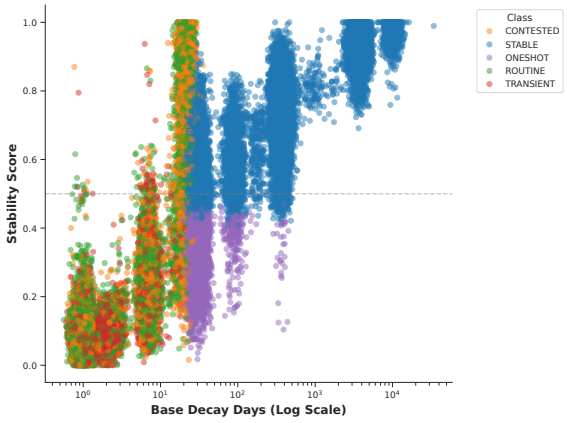

Figure 1: Distribution of the five memory classes by ground-truth stability score and validity horizon.

2.3 Memory Dynamics

The benchmark requirements are informed by memory behaviors emphasized across prior work on long-term memory and personalization (Maharana et al., 2024; Wu et al., 2025; Jiang et al., 2025; Uddin et al., 2026; McClelland et al., 1995; Park & Bak, 2024). Following generation, the facts are grouped into five management classes based on their annotated validity horizon, stability, reinforcement history, and contradictions:

1. **STABLE (26%):** Long-lived (90–3650 days), typically introduced once (e.g., profession). Requires durable retention over long horizons.
2. **ROUTINE (42%):** Short-lived (2–21 days) observations that repeatedly recur with the same value. Requires evidence to accumulate rather than treating each observation independently.
3. **CONTESTED (20%):** Information whose value is revised over time (e.g., current project). Requires updating the belief state to the latest value and superseding previous state.
4. **TRANSIENT (7%):** Very short-lived (1–10 days), unrepeated information (e.g., today's lunch). Requires expiration and eventual abstention even without a subsequent contradiction.
5. **ONESHOT (4%):** Unrepeated information with an intermediate validity horizon (15–90 days). Requires temporary retention without either immediate loss or indefinite lifespans.

2.4 Query Evaluation

We evaluate management using queries placed at controlled points in each fact's lifecycle. A *retrieval query (VAL)* occurs while the fact is valid and is scored on recovering the current value. An

[1]We also provide naturalistic, non-templated versions of all inputs for evaluating retrieval-based methods. Appendix D uses these to run a small study showing that natural-language retrieval can be trained over the same substrate without modifying memory management.

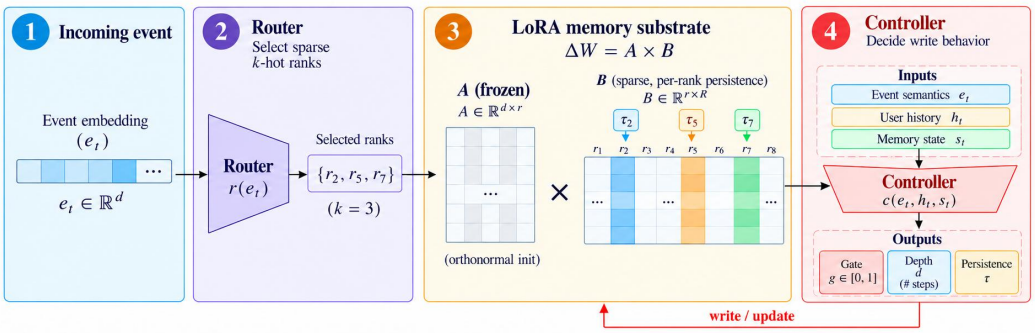

Figure 2: **Overview of the parametric memory management architecture.** An incoming user memory is mapped by the router to its storage in the LoRA adapter, consisting of a k-hot set of ranks in the up-projection (B) matrix. Based on the available online features including features from these ranks, the controller decides whether and how the information should be memorized.

abstention query (ABS) occurs after expiry and is scored on answering *Unknown*. For CONTESTED facts, queries target the currently valid value. Repeated ROUTINE events test whether consistent observations accumulate into stronger retention. From Figure 1, we can see that some classes have distinctive ground-truth patterns (STABLE vs TRANSIENT) but others can be overlapping (ROUTINE vs CONTESTED vs TRANSIENT). A valid management policy must therefore identify the user-level signals that segregate overlapping classes. Per-fact query budgets are reweighted to ensure that important but less frequent behaviors, such as expiration, receive sufficient evaluation mass despite write imbalances (Appendix C). The resulting query distribution is therefore designed as a controlled stress test of memory-management capabilities rather than an estimate of natural query frequency. Each class isolates a distinct challenge: long-horizon retention (STABLE), reinforcement accumulation (ROUTINE), belief revision (CONTESTED), principled expiration (TRANSIENT), and intermediate-horizon retention (ONESHOT). We report class-balanced (CB) accuracy so that performance is not dominated by the most frequent lifecycle class.

3 Memory Substrate

Prior work shows that LoRA adapters can serve as compact parametric stores for information (Xu et al., 2026; Back et al., 2026). Standard LoRA, however, is not well suited to lifelong memory management as consecutive writes update the same shared parameter space, causing interference between unrelated memories, while information written once has no mechanism to expire selectively (Yang et al., 2024). We therefore design a fixed-capacity substrate on a Qwen 3 1.7B backbone with three properties required for parametric memory management in the attached LoRA adapter: **isolated writes**, **controllable lifespans**, and **calibrated expiration**. Together, these provide the memory dynamics over which the management policy in §4 operates (Figure 2).

3.1 Sparse Combinatorial Memory Locations

We treat the R rank columns of the adapter's B matrix as an addressable memory bank. Each write is routed to a k-hot subset of columns while all others remain frozen and retrieval loads the same subset (Zou et al., 2025). The A matrix is orthonormally initialized and frozen throughout, as freezing A yields a more predictable, approximately proportional relationship between write depth and memorization strength (Zhang et al., 2026). With $R = 32$ and $k = 2$ as our main configuration, this yields $\binom{32}{2} = 496$ addressable locations from 32 shared basis columns. Larger k raises recall at the expense of abstention (Appendix A.1). This value-abstain trade-off dictates the limits of the management policies analyzed in §4.

This routing provides two core properties. First, **interference is bounded** as writes to disjoint masks share no gradient path, and contention scales with mask overlap $|m_1 \cap m_2|$ only. Second, **repeated writes accumulate** as five writes at depth $d = 20$ (20 gradient descent steps at fixed learning rate) achieve comparable retention to a single $d = 100$ write (Appendix A.2), allowing policies to reinforce facts via incremental updates. A key component of this architecture is the router that assigns related facts and queries to the same k-hot ranks for reliable updates and retrieval. We evaluate two routers. Enabled by our templated setting, the *perfect router* deterministically hashes templates to masks, upper-bounding routing. The *learned router* assigns slots using cosine similarity

on the fact's whitened embedding (Su et al., 2021) with existing slots, allocating new masks when similarity is low (Appendix A). Routing is a separable, independently tuned module. By evaluating both perfect and learned variants, we decouple the impact of storage from management.

3.2 Controllable Temporal Lifespans

To support selective expiration, each rank column i carries an independent decay constant τ_i. Between events separated by Δt days, $B[:, i] \leftarrow B[:, i] \cdot \exp(-\Delta t/\tau_i)$. Each write overwrites τ_i for its columns, letting the policy assign different lifespan horizons to different information. We discretize τ into five temporal scales using empirical decay timelines, from short-lived to effectively permanent (values in §4.1). Per-location decay is structurally necessary because stable and transient information coexist in the same fixed-capacity adapter. A single global rate either expires stable facts prematurely or preserves transient ones indefinitely (validated in §4.4, where a global τ reduces abstention on all volatile classes). Independent decay makes forgetting a controllable operation where information fades and its capacity is reclaimed without an explicit unlearning step. However, when two facts are routed to overlapping columns, the older fact's decay rate is overwritten by the newer fact's decay rate. We analyze the impact of this shared-column limitation in §4.5.

3.3 Calibrated Retrieval and Abstention

A memory substrate must also represent absence. A weak or expired trace should produce abstention rather than a hallucinated value. Base models tend to fabricate values from weak parametric traces, and prior work treats this behavior as an intrinsic threshold of parametric memorization (Xu et al., 2026). We show that abstention from weak traces is instead a resolvable, calibration problem.

A separate rank-32 *retrieval adapter* operates jointly with the memory adapter during inference, translating memory traces into expected outputs. Instead of memorizing facts itself, the retrieval adapter is trained to read the underlying substrate's state using depth-conditioned labels. If a routed slot is empty, it is trained to output `Unknown`. If a fact was written to the memory adapter at depth d, the retrieval adapter's target is `Unknown` with probability $p_{\text{unk}}(d)$ and the correct value otherwise (e.g. $p_{\text{unk}}(20) = 0.9$ and $p_{\text{unk}}(150) = 0$).

The shape of the monotonically decreasing curve $p_{\text{unk}}(d)$ (Appendix A.3) governs behavior under weak writes. A curve that drops too soon may retrieve hallucinated values from weak traces, while one that drops too late may collapse all queries to abstention. This probabilistic mapping yields a graded storage mechanism, rather than a binary retrievable threshold, giving the controller meaningful control over retention strength via d. Crucially, because the retrieval adapter uses the state of the substrate to generate answers rather than the input, retrieval is separated from memory management. The same substrate can be paired with alternative adapters, such as one trained for free-form natural language instead of our templates, without altering the underlying mechanics. (Appendix D).

3.4 Why Continual Learning Is Insufficient

Do these properties emerge from applying standard continual learning (CL) to a shared LoRA store? We test EWC (Kirkpatrick et al., 2017) and PCGrad with sequential fixed-depth LoRA writes, selecting $d = 15$ (optimal selection from $d = 10, 15, 20, 40, 60$). Both reduce interference by protecting previously written information but provide no mechanism for selective expiration. Consequently, every CL baseline achieves 0.00 ABS despite PCGrad reaching 0.82 VAL (Table 2). Adding uniform exponential decay changes accuracy by < 0.01, since dense shared parameters cannot assign different lifespans to different memories. Our substrate resolves this structurally as specific ranks decay to physically represent forgetting, which trigger the retrieval adapter to abstain. Sparse locations, independent decay, and calibrated retrieval provide this control.

Method	CB ↑	VAL ↑	ABS ↑
Naive LoRA	0.23	0.51	0.00
EWC	0.22	0.50	0.00
PCGrad	0.42	**0.82**	0.00
Ours (PMM-PR)	**0.60**	0.68	**0.59**

Table 2: CL protects stored information but cannot represent selective expiration. Per-class breakdowns are in the Appendix A.4.

4 LEARNED MEMORY MANAGEMENT

The substrate defines the storage of parametric memory. Here, we investigate (1) what actions are required to manage this substrate, (2) whether the information available online is sufficient to infer effective management decisions, and (3) what trade-offs remain even under effective management due to the substrate's finite capacity.

4.1 MEMORY MANAGEMENT ACTIONS

For each incoming event, the controller emits an action (g, d, τ):

- **Gate** ($g \in 0, 1$): whether to write the event at all. Declining leaves the substrate unchanged.
- **Depth** ($d \in 0, 20, 30, 40, 60, 80, 110, 150$ SGD steps): how strongly to write. Higher d yields stronger retention but a larger gradient footprint on shared mask columns.
- **Lifespan** ($\tau \in 25, 100, 400, 1600, 20000$): the decay timescale assigned to the routed columns, controlling how long the resulting trace persists.

The three actions are predicted by independent discretized heads, allowing each to be conditioned and bias-initialized separately (§4.3). Discretization also constrains the policy search space and simplifies optimization. We find two design choices to be important. First, the action space distinguishes *rejecting* information ($g = 0$) from *briefly retaining* it through a shallow write with short lifespan. Different volatile events may require either behavior, motivating independent gate and lifespan controls. Second, routing is separated from the management policy. The router in Section 3.1 determines which ranks an event is written in, while the controller determines how/whether it is stored through gate, depth, and lifespan. We evaluate both perfect and learned routing to separate allocation errors from management errors.

4.2 STATE AND POLICY ARCHITECTURE

The 2M-parameter controller ($< 0.4\%$ of the base model) conditions on three sources of online information: (1) the **incoming event** represented by its semantic embedding, (2) **trajectory history** represented by a GRU state (Cho et al., 2014), and (3) **substrate state**, consisting of per-rank ages (time since last write for each rank), per-rank norms (to capture stored information), and token-level prediction signals from a forward pass that indicate whether the incoming event reinforces or contradicts the existing memory. A shared neural trunk feeds independent decision heads for gate, depth, and lifespan (Appendix B.4).

Persistence cannot be determined from event content alone. Because identical facts exhibit different stabilities across users, relying solely on semantics leads to a prediction ceiling of only $R^2 = 0.36$ (Appendix B.13). The controller addresses this ambiguity by grounding its decisions in the user's trajectory history. Finally, to stabilize early training, the action heads are initialized with empirical biases derived from an expert policy.

4.3 LEARNING FROM A METADATA ORACLE

We train the controller by imitation from a manually configured *causal metadata oracle*. The oracle observes the same online history as the controller together with ground-truth lifecycle metadata (stability, time to decay, and reinforcement status) and computes a target write depth based on a deficit. The deficit is the estimated depth required for a fact to remain retrievable over its target horizon, minus any depth already accumulated through prior writes, capped at 150. The oracle has no access to future events, but receives lifecycle metadata that the learned controller must infer from its observable state features. Crucially, this supervision adapts to the student's actual substrate state under DAgger (Ross et al., 2011). When the student deviates from the oracle, the target depth is recomputed from what the student has actually written rather than assuming ideal previous actions. Replacing this state-aware supervision with fixed target depths derived solely from ground-truth metadata reduces class-balanced accuracy from 0.56 to 0.49 (Appendix B.7).

Training. The training pipeline has three stages: (A) stability-head MSE training, (B) behavior cloning against the oracle via DAgger, and (C) optional PPO refinement (Schulman et al., 2017) (Appendix B.15). Stage B reaches near-oracle performance and is our primary configuration. We compare our parametric memory management (PMM) controllers trained with perfect routing (PMM-PR) and learned routing (PMM-LR) to separate the learned policy impact from allocation quality.

System	CB	STABLE	ROUTINE	CONTESTED	TRANSIENT	ONESHOT
Metadata oracle	$0.65 \pm .02$	$0.56 \pm .03$	$0.72 \pm .03$	$0.51 \pm .04$	$0.77 \pm .03$	$0.68 \pm .03$
PCGrad (best CL, $d = 15$)	$0.42 \pm .01$	$0.60 \pm .03$	$0.66 \pm .03$	$0.41 \pm .04$	$0.13 \pm .03$	$0.31 \pm .03$
RAG (1.7B)	$0.50 \pm .01$	$0.30 \pm .03$	$0.42 \pm .03$	$0.55 \pm .04$	$0.75 \pm .03$	$0.49 \pm .03$
RAG-MM (1.7B)	$0.50 \pm .01$	$0.41 \pm .03$	$0.45 \pm .04$	$0.48 \pm .04$	$0.68 \pm .03$	$0.47 \pm .03$
RAG-MM (122B)	$0.53 \pm .01$	$0.47 \pm .03$	$0.69 \pm .03$	$0.47 \pm .04$	$0.33 \pm .03$	$0.67 \pm .03$
Unmanaged $d = 20$	$0.45 \pm .01$	$0.56 \pm .03$	$0.63 \pm .03$	$0.43 \pm .04$	$0.26 \pm .03$	$0.38 \pm .03$
Unmanaged $d = 80$	$0.50 \pm .01$	$0.65 \pm .03$	$0.69 \pm .03$	$0.51 \pm .04$	$0.11 \pm .03$	$0.54 \pm .03$
PMM-LR	$\mathbf{0.56 \pm .01}$	$0.40 \pm .03$	$0.76 \pm .03$	$0.56 \pm .04$	$0.44 \pm .04$	$0.65 \pm .03$
PMM-PR	$0.60 \pm .01$	$0.39 \pm .03$	$0.75 \pm .03$	$0.57 \pm .04$	$0.59 \pm .04$	$0.68 \pm .03$
No per-rank τ (PMM-PR)	$0.48 \pm .01$	$0.57 \pm .03$	$0.70 \pm .03$	$0.44 \pm .04$	$0.24 \pm .03$	$0.47 \pm .03$

Table 3: Evaluation results with Wilson 95% CIs on per-class total accuracy ($n = 15$ held-out users, 4584 queries). RAG-MM refers to RAG with a write-time memory-management architecture (Appendix B.3). Naive/EWC continual-learning variants (≤ 0.23 CB), oracle-metadata RAG upper bounds, and no-τ unmanaged baselines are reported in Appendix B.6.

4.4 Results

We evaluate on 15 held-out user trajectories (~ 600 events per user) with sequential LoRA writes and queries. Results are reported per lifecycle class and per query type (VAL, ABS), aggregated using class-balanced (CB) accuracy. Table 3 answers three questions. First, can conventional parametric memorization solve the lifecycle problem? No, CL baselines and unmanaged-write on our substrate retrieve values well but fail to expire outdated information. Second, is lifecycle management learnable? Yes, PMM-LR recovers 86% of the metadata-oracle ceiling. Third, what limits performance? The fully learned PMM-LR reaches 0.56 CB accuracy, while perfect routing raises performance to 0.60 against a metadata-oracle ceiling of 0.65. This hierarchy separates distinct sources of error. The PMM-LR-to-PMM-PR gap reflects semantic allocation, the PMM-PR-to-oracle gap primarily reflects lifecycle inference, and the oracle ceiling reflects limitations of the substrate itself, including shared-column interference between distinct memories and τ assignment (analyzed below). Improving routing can close the first gap, richer online management signals can close the second, and changes to allocation or capacity are required to raise the substrate ceiling.

Both controllers outperform all CL and unmanaged-write baselines, as well as the retrieval-based memory managers tested at compute parity. PMM-LR reaches 0.56 versus 0.50 for compute-matched RAG-MM and also exceeds RAG-MM with a substantially larger 122B management model (0.53). Retrieval surpasses our controllers when its management pass is replaced with ground-truth lifecycle information (0.64 at 1.7B and 0.82 at 122B §4.5), highlighting the importance of the management signal independent of the underlying memory substrate. Once information is written, queries require no retrieval over an external history and no context growth with trajectory length.

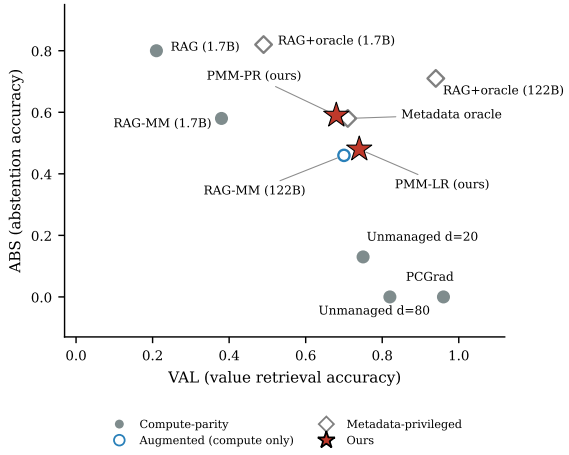

Figure 3: Pareto positions in value-retrieval (VAL) versus abstention (ABS) accuracy. Four regimes emerge: (1) continual learning and unmanaged writes at high VAL and near-zero ABS, (2) compute-parity retrieval at high ABS and low VAL, (3) augmented retrieval using greater compute or privileged lifecycle information reaching the top-right, and (4) our controllers occupying a balanced regime that no compute-parity baseline reaches. Breakdown in Table 10.

4.5 Per-Class Trade-offs and Failure Modes

Memorization without management fails at abstention. Fixed-depth writes demonstrate a clear trend. At $d = 20$, VAL ranges from 0.60-0.90 across classes while ABS remains only 0.01-0.22. Increasing depth to $d = 80$ raises VAL to 0.85-1.00 while driving ABS to ≤ 0.01 (per-class VAL/ABS splits in Appendix B.1). Adding τ without event-specific management does not improve this trade-off (Appendix B.2).

Retrieval-based memory also requires explicit management. Compute-matched simple RAG fails due to frequently defaulting to `Unknown` under ambiguous context (ABS 0.75–0.87) while rarely committing to values (VAL 0.16–0.30 on most classes). RAG-MM's write-time management pass—resolving reinforcement, contradiction, and expiry as events arrive (Appendix B.3)—shifts this Pareto position. Replacing the learned management pass with ground-truth lifecycle information raises RAG-MM to 0.64 at 1.7B and 0.82 at 122B (Appendix B.6), providing an upper bound when management is ideal and not inferred (Figure 3).

Management matters where events require different lifespans. Our controllers occupy a balanced regime that other compute-parity baseline do not reach. For TRANSIENT facts, no unmanaged fixed-depth policy exceeds 0.26 accuracy because writing them strongly enough for immediate retrieval causes them to remain accessible after expiry. PMM-LR and PMM-PR reach 0.44 and 0.59, respectively, by assigning short lifespans to volatile information. Conversely, on STABLE facts, unmanaged $d = 80$ reaches 0.65 while PMM reaches 0.39–0.40, pairing moderate VAL (0.33–0.34) with strong ABS (0.78–0.81).

The STABLE gap is due to τ overwrites on shared columns. Failures in retrieval of STABLE facts occur primarily at longer query horizons and under greater foreign-write interference, rather than from policy failures like declined writes or insufficient initial depth (Appendix B.14). Because each write overwrites τ on the columns it touches, a STABLE fact can inherit the shorter timescale of a later volatile write sharing the same column over its long horizon. Preserving the maximum τ recovers $\sim$13 points of STABLE VAL but collapses ABS on volatile classes, reducing CB accuracy to 0.45. This exposes a lifespan-compression trade-off in the shared substrate. Reinforcement partially mitigates the effect as STABLE facts with $3+$ writes achieve 0.68 retrieval success as compared to 0.51 for single-write facts. High STABLE retrieval performance can be achieved by routing them to non-overlapping ranks, at the cost of compression.

Per-rank τ enables joint optimization. Removing per-rank lifespan control and forcing a global τ drops PMM-PR from 0.60 to 0.48. Without the ability to selectively expire information, the controller maintains retrieval only by sacrificing abstention. VAL remains ≥ 0.79 across volatile classes, while TRANSIENT ABS falls to 0.09, ROUTINE ABS to 0.06, and CONTESTED ABS to 0.04. This establishes independent lifespan control as a necessary mechanism for supporting both retention and expiration in this substrate.

Ablations. Table 4 reports input-feature ablations for PMM-PR. Removing semantic features or configured priors substantially reduces aggregate accuracy (0.47 and 0.52, respectively), confirming that both provide non-redundant signals during training. History and substrate state are partially redundant because both reflect previous observations (through trajectory or through substrate state impact). However, class-specific failures highlight the distinct mechanistic role of each feature. Without trajectory history, ONESHOT VAL drops by 24 points because these first-time events lack a pre-existing trace to provide a redundant signal. Conversely, without substrate state, ROUTINE ABS drops by 9 points. Blind to how strongly a recurring fact is already embedded, the controller over-relies on historical frequency and over-writes the trace, preventing it from properly expiring. These results show that effective lifecycle management depends on event semantics, the user's trajectory, and the current state of memory. Full per-class results are provided in Appendix B.16.

Variant	CB acc	Notable effect
PMM-PR (full)	0.60	baseline
No history	0.58	ONESHOT VAL -24
No substrate	0.58	ROUTINE ABS -9
No priors	0.52	overall -8pt
No semantic	0.47	overall -13pt

Table 4: Ablations on controller input features.

5 Related Work

5.1 Parametric Memory and Lifelong Knowledge Editing

A growing body of work stores information in model parameters or latent states rather than external text (Sun et al., 2025; Ba et al., 2016). Recent systems divide into latent-state memory, maintaining pools of hidden-state memory tokens (Wang et al., 2024; 2025a), and parametric memory, writing into weights through context distillation (Wang et al., 2025b), dedicated memory modules (Berges

et al., 2024), or test-time weight updates (Behrouz et al., 2024). In parallel, lifelong knowledge editing develops efficient, localized modification of model knowledge (Meng et al., 2023; Hartvigsen et al., 2023; Yu et al., 2023). Both lines study whether information can be written, retrieved, or updated while preserving unrelated knowledge. We instead study memory lifecycle management which explores how strongly information should be stored, how long it should persist, and when new observations should reinforce or replace existing beliefs.

5.2 Sparse and Modular Parameter Updates

Sparse and modular weight updates reduce interference during updates. O-LoRA (Wang et al., 2023) uses near-orthogonal task subspaces, N-LoRA (Yang et al., 2024), and FlyLoRA (Zou et al., 2025) allocate sparse adapter ranks to increase capacity and limit overlap. Our substrate similarly uses sparse combinations of adapter ranks, but as reusable memory locations rather than task-specific modules. Further, parametric overlap is not uniformly undesirable as unrelated writes should stay isolated, while overlap between related information supports reinforcement and overwrite. We further provide a controlled temporal decay for each rank, making persistence an explicit property of the memory rather than a byproduct of the memorization.

5.3 Memory Management and Continual Learning

To our knowledge, no prior work attempts to govern the parametric memory lifecycle—retention, expiration, reinforcement, and overwrite—as a learned management problem. Retrieval-based memory management systems organize external stores through explicit hierarchies and paging (Packer et al., 2024; Kang et al., 2025), learned memory operations (Yu et al., 2026; Yan et al., 2026), or importance- and lifespan-based retention (Park & Bak, 2024), demonstrating that what is retained or forgotten can be explicitly managed. These managers, however, operate over explicit entries that can be directly added, updated, or deleted. Our controller instead governs the formation and lifespan of information encoded in parameters—deciding whether, how strongly, and for how long to write—so retention and forgetting emerge from the dynamics of the substrate itself.

Continual learning (Zeng et al., 2019; Yu et al., 2020; Saha et al., 2021) addresses the complementary problem of acquiring new knowledge while minimizing interference with the old. Our experiments (§3.4) show memorization alone is insufficient for long-horizon memory, where some information must persist while other information must become inaccessible.

6 Conclusion

We present parametric memorization and parametric memory management as distinct problems, often conflated by prior work. Long-horizon memory requires deciding what to retain, reinforce, overwrite, or let expire. Existing parametric memory mechanisms do not explicitly expose or control these lifecycle decisions. Separating memory into its substrate dynamics—a fixed-capacity substrate with sparse, addressable locations and controllable temporal decay—and its policy—a lightweight controller governing the substrate through a defined action space—enables the system to balance retrieval of valid information against abstention on information that should no longer be accessible. Our controlled environment supports two central findings: (1) a fixed-capacity parametric substrate can support varying memory lifecycles, and (2) the decisions required to manage those lifecycles can be inferred from online observations.

Limitations. Our evaluation uses a templated interface to isolate lifecycle management from natural-language variation (analyzed in Appendix D). We also assume memory identification, determining what information constitutes a memory event, is given, and our controlled trajectories may not cover all real-world user dynamics.

Future work. The learned-routing, controller, and oracle gaps motivate better semantic addressing, richer write-time signals, and improved capacity allocation. RL over long-term memory utility may further improve the controller, while non-overlapping allocation and query-time reinforcement may raise the substrate ceiling.

Overall, our results suggest that reliable parametric memory requires progress beyond how models memorize. It requires controlling how information evolves after it has been written. By showing that a 2M-parameter policy operating on online signals recovers 86-92% of a metadata-privileged oracle, we establish parametric memory management as a distinct and learnable problem and provide a foundation for lifecycle-aware memory in model parameters.

AI USE STATEMENT

In this work, we used generative AI for synthetic data generation, code generation, as well as for editing the draft and figures. We did not use generative AI tools for other research activities. All AI-assisted code was reviewed and tested for correctness by the authors, and all AI-assisted text edits were reviewed and approved before inclusion. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our findings, we extensively document all components of our dataset, memory substrate, learned controller, and baselines throughout the main text and appendices:

- **Dataset:** We will publicly release the complete 909-user trajectory benchmark, including chronological event streams, ground-truth lifecycle metadata, and naturalistic text variants. The five-stage generation pipeline is detailed in §2.2, with temporal motif distributions and dataset statistics provided in Appendix C.
- **Memory Substrate:** The physics of the sparse combinatorial substrate, including routing mechanics, bounds on gradient interference, and the exact depth-conditioned target probabilities for the retrieval adapter, are formalized in §3 and Appendix A.
- **Learned Controller:** To enable independent replication of the memory manager, we mathematically define the state representation engine and the action policy architecture in Appendix B.4. Furthermore, we provide the exact deficit-based expert policy used for DAgger supervision (Algorithm 1), along with the multi-task loss formulation, training schedules, bias initializations, and a comprehensive hyperparameter table in Appendix B.
- **Baselines:** The implementation details and hyperparameters for the continual learning baselines (EWC, PCGrad) are explicitly defined in Appendix A.4. Additionally, the exact system prompts and LLM management rules for the retrieval-based memory variants (RAG, RAG-MM, RAG+oracle) are documented in Appendix B.3.

REFERENCES

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.1, knowledge storage and extraction, 2024. URL https://arxiv.org/abs/2309.14316.

Jimmy Ba, Geoffrey Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past, 2016. URL https://arxiv.org/abs/1610.06258.

Seungju Back, Dongwoo Lee, Naun Kang, Taehee Lee, S. K. Hong, Youngjune Gwon, and Sungjin Ahn. Understanding lora as knowledge memory: An empirical analysis, 2026. URL https://arxiv.org/abs/2603.01097.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time, 2024. URL https://arxiv.org/abs/2501.00663.

Vincent-Pierre Berges, Barlas Oğuz, Daniel Haziza, Wen tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale, 2024. URL https://arxiv.org/abs/2412.09764.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014. URL https://arxiv.org/abs/1406.1078.

Tianyu Ding, Aditya Nannapaneni, Bingfan Liu, and Ling Zhang. Always-onagents:a survey of persistent memory, state, and governance in llmagents, 2026. URL https://arxiv.org/abs/2606.30306.

Hermann Ebbinghaus. *Memory: A Contribution to Experimental Psychology*. Duncker & Humblot, Leipzig, Germany, 1885. English translation published 1913 by Teachers College, Columbia University.

Ruxi Gu, Zhenliang Zhang, and Wei Wang. Forgetbench: Benchmarking forgetting dynamics of long-term parametric memory in language models, 2026. URL https://arxiv.org/abs/2607.26455.

Thomas Hartvigsen, Swami Sankaranarayanan, Hamid Palangi, Yoon Kim, and Marzyeh Ghassemi. Aging with grace: Lifelong model editing with discrete key-value adaptors, 2023. URL https://arxiv.org/abs/2211.11031.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

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.

Bowen Jiang, Zhuoqun Hao, Young-Min Cho, Bryan Li, Yuan Yuan, Sihao Chen, Lyle Ungar, Camillo J. Taylor, and Dan Roth. Know me, respond to me: Benchmarking llms for dynamic user profiling and personalized responses at scale, 2025. URL https://arxiv.org/abs/2504.14225.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent, 2025. URL https://arxiv.org/abs/2506.06326.

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.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021. URL https://arxiv.org/abs/2005.11401.

Jiahong Liu, Zexuan Qiu, Zhongyang Li, Quanyu Dai, Wenhao Yu, Jieming Zhu, Minda Hu, Menglin Yang, Tat-Seng Chua, and Irwin King. A survey of personalized large language models: Progress and future directions, 2025. URL https://arxiv.org/abs/2502.11528.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents, 2024. URL https://arxiv.org/abs/2402.17753.

James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102(3):419–457, 1995.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pp. 109–165. Academic Press, 1989. doi: https://doi.org/10.1016/S0079-7421(08)60536-8. URL https://www.sciencedirect.com/science/article/pii/S0079742108605368.

Nick Mecklenburg, Yiyou Lin, Xiaoxiao Li, Daniel Holstein, Leonardo Nunes, Sara Malvar, Bruno Silva, Ranveer Chandra, Vijay Aski, Pavan Kumar Reddy Yannam, Tolga Aktas, and Todd Hendry. Injecting new knowledge into large language models via supervised fine-tuning, 2024. URL https://arxiv.org/abs/2404.00213.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt, 2023. URL https://arxiv.org/abs/2202.05262.

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.

Sangjun Park and JinYeong Bak. Memoria: Resolving fateful forgetting problem through human-inspired memory architecture, 2024. URL https://arxiv.org/abs/2310.03052.

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2011. URL https://arxiv.org/abs/1011.0686.

Gobinda Saha, Isha Garg, and Kaushik Roy. Gradient projection memory for continual learning, 2021. URL https://arxiv.org/abs/2103.09762.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Jianlin Su, Jiarun Cao, Weijie Liu, and Yangyiwen Ou. Whitening sentence representations for better semantics and faster retrieval, 2021. URL https://arxiv.org/abs/2103.15316.

Weihang Su, Yichen Tang, Qingyao Ai, Junxi Yan, Changyue Wang, Hongning Wang, Ziyi Ye, Yujia Zhou, and Yiqun Liu. Parametric retrieval augmented generation, 2025. URL https://arxiv.org/abs/2501.15915.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): Rnns with expressive hidden states, 2025. URL https://arxiv.org/abs/2407.04620.

Md Nayem Uddin, Kumar Shubham, Eduardo Blanco, Chitta Baral, and Gengyu Wang. From recall to forgetting: Benchmarking long-term memory for personalized agents, 2026. URL https://arxiv.org/abs/2604.20006.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. Orthogonal subspace learning for language model continual learning, 2023. URL https://arxiv.org/abs/2310.14152.

Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. Memoryllm: Towards self-updatable large language models, 2024. URL https://arxiv.org/abs/2402.04624.

Yu Wang, Dmitry Krotov, Yuanzhe Hu, Yifan Gao, Wangchunshu Zhou, Julian McAuley, Dan Gutfreund, Rogerio Feris, and Zexue He. M+: Extending memoryllm with scalable long-term memory, 2025a. URL https://arxiv.org/abs/2502.00592.

Yu Wang, Xinshuang Liu, Xiusi Chen, Sean O'Brien, Junda Wu, and Julian McAuley. Self-updatable large language models by integrating context into model parameters, 2025b. URL https://arxiv.org/abs/2410.00487.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory, 2025. URL https://arxiv.org/abs/2410.10813.

Ziwen Xu, Haiwen Hong, Linsong Yu, Benglei Cui, Longtao Huang, Hui Xue, and Ningyu Zhang. How lora remembers? a parametric memory law for llm finetuning, 2026. URL https://arxiv.org/abs/2605.30260.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning, 2026. URL https://arxiv.org/abs/2508.19828.

Shuo Yang, Kun-Peng Ning, Yu-Yang Liu, Jia-Yu Yao, Yong-Hong Tian, Yi-Bing Song, and Li Yuan. Is parameter collision hindering continual learning in llms?, 2024. URL https://arxiv.org/abs/2410.10179.

Lang Yu, Qin Chen, Jie Zhou, and Liang He. Melo: Enhancing model editing with neuron-indexed dynamic lora, 2023. URL https://arxiv.org/abs/2312.11795.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning, 2020. URL https://arxiv.org/abs/2001.06782.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents, 2026. URL https://arxiv.org/abs/2601.01885.

Guanxiong Zeng, Yang Chen, Bo Cui, and Shan Yu. Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 1(8):364–372, August 2019. ISSN 2522-5839. doi: 10.1038/s42256-019-0080-x. URL http://dx.doi.org/10.1038/s42256-019-0080-x.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. Lora-fa: Efficient and effective low rank representation fine-tuning, 2026. URL https://arxiv.org/abs/2308.03303.

Heming Zou, Yunliang Zang, Wutong Xu, Yao Zhu, and Xiangyang Ji. Flylora: Boosting task decoupling and parameter efficiency via implicit rank-wise mixture-of-experts, 2025. URL https://arxiv.org/abs/2510.08396.

A Substrate Details

This appendix expands the substrate measurements, routing mechanics, retrieval adapter training, and continual-learning comparison summarized in Section 3.

A.1 Rank and k Selection Data

Table 5 shows the full metadata oracle selection across R and k with per-class VAL/ABS breakdown. All configurations use real substrate with per-rank τ and each measurement is over 5 held-out users (1452 queries).

	STABLE		ROUTINE		CONTESTED		TRANSIENT		ONESHOT		
Config	VAL	ABS	VAL	ABS	VAL	ABS	VAL	ABS	VAL	ABS	CB
Fixed $R = 32$, varying k											
$k = 1$	0.58	0.70	0.33	0.83	0.42	0.53	0.26	0.89	0.62	0.77	0.61
$k = 2$	0.64	0.70	0.76	0.74	0.85	0.27	0.46	0.80	0.97	0.46	**0.69**
$k = 4$	0.57	0.75	0.97	0.58	1.00	0.15	0.92	0.41	0.97	0.62	0.66
$k = 8$	0.49	0.75	0.95	0.40	1.00	0.09	0.97	0.21	0.92	0.48	0.57
$k = 16$	0.36	0.75	0.90	0.21	0.98	0.06	0.97	0.09	0.75	0.33	0.48
Fixed $k = 2$, varying R											
$R = 16$	0.52	0.82	0.81	0.67	0.91	0.23	0.67	0.63	0.85	0.62	0.65
$R = 32$	0.64	0.70	0.76	0.74	0.85	0.27	0.46	0.80	0.97	0.46	0.69
$R = 64$	0.73	0.58	0.67	0.75	0.73	0.27	0.41	0.91	0.96	0.59	**0.70**
$R = 128$	0.82	0.67	0.61	0.79	0.68	0.23	0.13	0.93	1.00	0.57	0.70

Table 5: Full metadata oracle selection across substrate rank R and mask size k, with VAL/ABS split per class. The value-abstain trade-off is visible in both dimensions. Increasing k generally shifts volatile and revision-heavy classes toward higher VAL at the cost of ABS, while increasing R redistributes between classes without changing aggregate substantially above $R = 32$.

The per-class VAL/ABS decomposition surfaces two mechanisms:

Increasing k shifts the value-abstention trade-off. CONTESTED VAL climbs from 0.85 at $k = 2$ to 1.00 at $k = 4$ and above, but CONTESTED ABS falls from 0.27 to 0.06. The same pattern is observed for TRANSIENT (VAL $0.46 \rightarrow 0.97$, ABS $0.80 \rightarrow 0.09$). Larger k means each write touches more columns, so facts are more likely to persist under any query (raising VAL) but also less likely to expire correctly (lowering ABS). At $k = 16$, the substrate is essentially always writing to overlapping columns and per-fact τ has been almost entirely overwritten.

Increasing R primarily improves STABLE. STABLE VAL climbs from 0.52 at $R = 16$ to 0.82 at $R = 128$, a 30-point improvement. This is exactly the mechanism we expect as more rank columns means each STABLE fact is less likely to share columns with subsequently-written short-τ facts. However, ROUTINE and CONTESTED VAL both drop as R grows because their routing dispersion increases (the router allocates similar facts to more distinct columns). The trade-off produces near-flat aggregate accuracy above $R = 32$, at which point $R = 32$ is preferred for its 2–4$\times$ smaller inner-loop compute per write.

Dynamic router mechanics. The dynamic router projects each fact's routing embedding through a learned whitening transform (fitted to dimension 256), then compares it by cosine similarity against the accumulated signature of every currently allocated mask. A mask's signature is maintained as an L_2-normalized exponential moving average ($\alpha = 0.5$) of the embeddings previously routed there. If the highest similarity exceeds a threshold of 0.85, the write is routed to that mask and its signature is updated, otherwise a new mask is allocated, with up to $\binom{R}{k}$ locations total.

A.2 Substrate Dynamics Measurements

Retrieval after first write, full curve. Table 6 extends Figure/Table 2 (main text) to all tested depths.

Depth d	10	15	20	30	40	60	80	110	150
Perfect router	0.00	0.00	0.00	0.05	0.13	0.61	0.78	0.89	0.97
Learnt router	0.00	0.00	0.01	0.15	0.49	0.84	0.90	0.94	0.98

Table 6: Retrieval after first write $P(\text{value})$ per depth.

Accumulation under repeated writes. Table 7 supports the constructive-accumulation claim of §3.1. Five writes at $d = 20$ ($P \geq 0.93$) substantially exceed a single write at the same depth ($P \approx 0$ at $d = 20$ per the table above), and approach the retention of a single deep write in the $d = 80$–110 range.

	$n = 1$	$n = 3$	$n = 5$	$n = 8$
$d = 20$ (perfect)	0.00	0.29	0.93	0.97
$d = 20$ (learnt)	0.01	0.45	0.95	0.98

Table 7: Cumulative retention $P(\text{value})$ after n writes at depth $d = 20$ with minimal intervening foreign traffic.

Contradict-primed overwrite. Table 8 characterizes overwrite behavior (§3.4): a contradicting value written at $d \geq 40$ dominates retrieval under both routing configurations.

Challenger depth	20	40	80	110
$P(v_2)$ perfect router	0.55	1.00	1.00	1.00
$P(v_2)$ learnt router	0.92	1.00	1.00	1.00

Table 8: Contradict overwrite. Initial fact is written at $d = 40$ with value v_1. The overwrite is a subsequent write with value v_2 at varying depths. $P(v_2)$ is the retrieval probability of the overwrite.

Priming and decay validation. To validate that per-rank exponential decay behaves as designed, we run a simple test. A fact is written at depth d with $\tau = 1000$, decayed for Δt days, and retrieved. Measured retention matches predicted $P(\text{cold}, d) \cdot \exp(-\Delta t/\tau)$ within measurement noise for $d \in \{40, 80, 110\}$ and $\Delta t \in \{10, 100, 300\}$ days.

Slot norm regression details. The slot norm regression takes as input a five-dimensional standardized feature vector: self steps, foreign steps, $\log(1 + \text{self steps})$, $\log(1 + \text{foreign steps})$, and time since last write. Cross-validated R^2 is 0.87 perfect routing and 0.75 on the learned router. The dynamic router's lower predictability reflects that its slot signatures drift with fact allocation, adding noise the regression cannot capture. This regression is what allows offline controller training to substitute predicted slot norm for a measured one (§4.2), at a $15\times$ speedup.

Forward pass test. Before each write, we compute $P(\text{value} \mid \text{current slot state}) - P(\text{value} \mid \text{clean slot})$This provides a substrate-derived signal of whether the incoming value is already represented in the routed slot. The difference is positive for reinforcement events and near zero for first-write events. AUC against the ground-truth reinforcement labels is 0.99 on both routers.

A.3 Retrieval Adapter Training

We train a rank-32 retrieval adapter jointly loaded with the memory adapter during both writes and reads. The training data is a mixed recipe: some facts get shallow writes ($d \in \{10, 15, 20\}$), some medium ($d \in \{30, 40\}$), some deep ($d \in \{60, 80\}$), some very deep ($d \in \{100, 130\}$), and some slots are left empty. For each written fact, we sample the target as `Unknown` with probability $p_{\text{unk}}(d)$ or the correct value otherwise.

Table 9 shows the final noise curve alongside the implied target retrieval probability. The constraint $p_{\text{unk}}(d) \leq 0.55$ prevents the retrieval from collapsing to always-abstain at any depth.

Depth d	10	20	30	40	60	80	110+
$p_{\text{unk}}(d)$	0.55	0.38	0.30	0.22	0.12	0.05	0.00
Implied $P(\text{value})$	0.10	0.20	0.32	0.45	0.70	0.85	0.95+

Table 9: Retrieval adapter noise curve. Each depth targets a specific $P(\text{value})$ during SFT.

A.4 Continual Learning Baseline Details

We evaluate two continual-learning methods against our substrate, all applied to LoRA writes at fixed depth $d = 15$ (chosen from $d \in \{15, 30, 60\}$ for each method). $d = 15$ maximized each method's aggregate accuracy. All CL baselines use the same base model and LoRA (rank 32 on q_{proj}, v_{proj}) but with fixed depth and no per-rank τ.

Naive fine-tuning. Direct backprop on the LoRA parameters at each write event, no regularization. Serves as the floor baseline for LoRA-based approaches.

EWC (Elastic Weight Consolidation). Adds a Fisher-weighted penalty term to write losses: $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{write}} + \lambda \sum_i F_i(\theta_i - \theta_i^*)^2$ where F_i is the diagonal Fisher information at parameter i estimated from the calibration data, θ_i^* is the pre-write parameter, and $\lambda = 1000$ (from (Kirkpatrick et al., 2017)). Fisher is updated with exponential moving average, $\alpha = 0.9$.

PCGrad (Projected Conflicting Gradients). Maintains a running memory of the $k = 5$ most recent write gradients. Each new write gradient is projected to be orthogonal to any stored gradient with which it has negative cosine similarity, following (Yu et al., 2020). This is designed to prevent new writes from destroying prior ones.

Retrieval adapter compatibility. CL baselines cannot use our retrieval adapter directly because the resoute was trained assuming the k-hot mask retrieval. For fair comparison, we retrained a compatible retrieval adapter for each CL baseline that reads from the full LoRA weight rather than a masked subset. This ensures accuracy differences reflect substrate design, not reading calibration mismatches.

B LEARNED MEMORY MANAGEMENT DETAILS

This appendix expands the controller architecture, expert policy, training details, per-class breakdowns, retrieval baseline architecture, and negative results summarized in Section 4.

B.1 FULL VAL/ABS BREAKDOWN

Table 10 shows the per-pattern accuracy split by target type (VAL versus ABS) for the systems in Table 3 and Appendix B.6.

	STABLE		ROUTINE		CONTESTED		TRANSIENT		ONESHOT	
System	VAL	ABS	VAL	ABS	VAL	ABS	VAL	ABS	VAL	ABS
Metadata oracle	0.53	0.74	0.75	0.65	0.84	0.24	0.52	0.81	0.93	0.46
Naive $d = 15$	0.17	0.00	0.72	0.00	0.63	0.00	0.76	0.00	0.26	0.00
EWC $d = 15$	0.19	0.00	0.73	0.00	0.64	0.00	0.76	0.01	0.19	0.00
PCGrad $d = 15$	0.70	0.00	0.96	0.00	0.89	0.00	0.91	0.00	0.64	0.00
Unmanaged $d = 20$	0.64	0.09	0.90	0.01	0.87	0.06	0.77	0.22	0.55	0.27
Unmanaged $d = 40$	0.81	0.00	0.99	0.00	0.99	0.01	0.96	0.02	0.92	0.03
Unmanaged $d = 80$	0.85	0.01	0.99	0.00	0.98	0.00	1.00	0.00	0.97	0.01
Unmanaged $d = 110$	0.83	0.00	1.00	0.00	1.00	0.00	1.00	0.00	0.97	0.00
RAG (1.7B)	0.22	0.76	0.27	0.75	0.16	0.87	0.26	0.84	0.16	0.79
RAG-MM (1.7B)	0.39	0.53	0.45	0.46	0.38	0.56	0.36	0.74	0.30	0.63
RAG-MM (122B)	0.44	0.65	0.81	0.42	0.75	0.24	0.94	0.22	0.55	0.78
RAG + oracle metadata (1.7B)	0.39	0.78	0.55	0.65	0.50	0.75	0.60	0.90	0.41	1.00
RAG + oracle metadata (122B)	0.86	0.77	0.84	0.54	1.00	0.53	1.00	0.75	1.00	0.97
PMM-PR (ours)	0.33	0.81	0.80	0.64	0.81	0.34	0.66	0.58	0.82	0.58
PMM-LR (ours)	0.34	0.78	0.88	0.47	0.93	0.21	0.83	0.35	0.71	0.60
No per-rank τ (PMM-PR)	0.55	0.68	0.96	0.06	0.87	0.04	0.83	0.09	0.79	0.22

Table 10: Per-pattern accuracy split into VAL and ABS query types. $n = 15$ held-out users.

B.2 UNMANAGED BASELINES WITHOUT τ

For completeness, we ran the unmanaged baseline depth selections both with per-rank τ (main table) and without τ (this table).

System	CB	STABLE	ROUTINE	CONTESTED	TRANSIENT	ONESHOT
$d = 20$	$0.45 \pm .01$	$0.54 \pm .03$	$0.62 \pm .03$	$0.41 \pm .04$	$0.30 \pm .03$	$0.40 \pm .03$
$d = 40$	$0.49 \pm .01$	$0.70 \pm .03$	$0.68 \pm .03$	$0.46 \pm .04$	$0.15 \pm .03$	$0.45 \pm .03$
$d = 80$	$0.50 \pm .01$	$0.73 \pm .03$	$0.68 \pm .03$	$0.45 \pm .04$	$0.15 \pm .03$	$0.47 \pm .03$
$d = 110$	$0.49 \pm .01$	$0.71 \pm .03$	$0.69 \pm .03$	$0.46 \pm .04$	$0.15 \pm .03$	$0.47 \pm .03$

Table 11: Unmanaged baselines without τ are nearly identical to their with-τ counterparts.

The near-identical aggregate numbers (all within ± 0.01 CI overlap) demonstrate an important point that τ alone is not what enables our controller's performance. The controller's benefit comes from combining per-fact τ policy with per-fact depth and gate decisions.

B.3 RETRIEVAL BASELINE DETAILS

All retrieval baselines share three components: an external text store containing the naturalistic realization of each event, a BGE-small embedder (33M parameters) for indexing and query-time retrieval, and a generator. Because the data is an event stream, no text chunking is required and facts are embedded directly as individual, timestamped strings (e.g., `[Clock: 0045] I cook dinner`). At query time, the system retrieves the top $k = 5$ most similar entries. The generator prompt explicitly instructs the model to read the retrieved timestamped entries and output `Unknown` if the memory does not clearly support an answer or appears outdated.

The variants differ only in how the store is maintained:

- **RAG:** the standard baseline. Events are appended to the store as they arrive and no management is applied. Retrieval at query time returns the top-k most similar entries.
- **RAG-MM:** the store is actively managed at write time by an LLM management pass. As each event arrives, the pass resolves its relationship to existing entries—merging reinforcements of the same fact, superseding contradicted values, expiring entries past their inferred validity, and deduplicating. It receives the new fact and retrieved candidates, and outputs a JSON decision to either ADD, UPDATE (soft-deleting the old entry), REINFORCE (merging and extending lifespan), or IGNORE. It also assigns an estimated decay in days (e.g., Transient: 7.0, Routine: 30.0, Stable: 3650.0).
- **RAG + oracle metadata:** identical to RAG-MM except the LLM management pass is replaced by the ground-truth lifecycle policy, using each event's generation-time metadata (validity horizon, contradiction and reinforcement status) to maintain the store perfectly. This removes lifecycle-management error while retaining the same retrieval and generation pipeline, providing an upper bound on retrieval-side memory management under privileged lifecycle information.

Generators. Compute-matched rows (labeled 1.7B) use Qwen3-1.7B, the same base model as our substrate. Large-generator rows (labeled 122B) use Qwen 3.5 122B A10B.

B.4 Controller Architecture Details

The learned management system is a 2M-parameter neural network consisting of two primary modules: a State Representation Engine (which maintains trajectory context and extracts semantic features) and the Action Policy (which predicts lifecycle management decisions). All internal representations use a hidden dimension of $d = 256$.

1. State Representation Engine The representation engine processes the raw outputs of the base language model into dense, controller-ready features. Let $x_t \in \mathbb{R}^{2048}$ be the last-token embedding of the incoming event template from the base model.

1. **Semantic Projection:** The raw embedding is normalized via a fitted input statistics layer and projected to a dense representation to decouple it from lexical noise: $z_t = \text{MLP}_{\text{sem}}(x_t) \in \mathbb{R}^{256}$.
2. **Trajectory GRU:** To capture user-specific behavioral context, a Gated Recurrent Unit (GRU) maintains a hidden state $h_t \in \mathbb{R}^{256}$. At each event, the GRU consumes the dense embedding z_t and the token-level forward pass result (probability of fact before writing) $p_{\text{hit}} \in [0, 1]$ (indicating if the model already knows the fact):

$$h_t = \text{GRUCell}(x_{\text{in}} = [z_t \oplus p_{\text{hit}}],\ h_{t-1})$$

2. Auxiliary State Heads Before computing actions, the network predicts the fact's structural properties using one auxiliary head attached to the representation engine. Because this property depend only on semantics (not the substrate), its input is detached from the computational graph to prevent auxiliary gradients from corrupting the shared representation trunk.

- **Stability Head:** A scalar prediction of the fact's true stability score, optimized via Mean Squared Error (MSE): $\hat{s}_t = \sigma(\text{Linear}(z_t \oplus h_t)) \in (0, 1)$.

3. Action Policy and Shared Trunk The main controller determines the action tuple $(\text{gate}, \text{depth}, \tau)$ by observing both the semantic trajectory and the state of the underlying LoRA substrate. The input to the shared policy trunk is the concatenation of six feature families:

$$s_{\text{policy}} = \left[z_t \oplus h_t \oplus A \oplus ||W_{\text{slot}}||_2 \oplus \hat{s}_t \oplus F_{\text{rank}} \oplus p_{\text{hit}} \right]$$

where $A \in \mathbb{R}^R$ are the rank ages, $||W_{\text{slot}}||_2$ is the predicted or measured L_2 norm of the active LoRA mask, and F_{rank} are the substrate interference features (foreign and self steps).

4. Decision Heads and Priors The policy state s_{policy} is passed through independent linear heads to produce the final action logits. The model utilizes empirical priors to stabilize training and guide the depth decision:

- **Gate Head ($g \in \{0, 1\}$):** Outputs a single logit l_{gate}. To prevent the gate from collapsing to 0 early in training, the logit is initialized with an empirical bias b_{gate} derived from the training set's marginal write rate: $P(\text{write}) = \sigma(l_{\text{gate}} + b_{\text{gate}})$.
- **Persistence Head ($\tau \in \{25, 100, 400, 1600, 20000\}$):** Outputs a 5-dimensional logit vector l_τ, shifted by empirical class frequency biases to account for the heavy skew toward shorter decay horizons.
- **Depth Head ($d \in \mathcal{D}_{>0}$):** Outputs a 7-dimensional logit vector l_{depth} over the available non-zero write depths $\{20, 30, 40, 60, 80, 110, 150\}$. The depth logits are directly augmented by the auxiliary head, acting as a learned prior.

  $$l_{\text{depth}}^{\text{final}} = l_{\text{depth}} + b_{\text{empirical}} + \alpha_1(\hat{s}_t V_{\text{stab}})$$

  where V_{stab} is a vector pushing high-stability predictions toward deeper bins.

B.5 Training Dynamics and Loss Formulation

Training a shared representation trunk alongside multiple action heads presents significant optimization challenges. To stabilize Behavioral Cloning (Phase 1) and ensure smooth convergence, we employ three specific training techniques.

1. Ordinal Soft Targets for Depth Classification Standard cross-entropy loss heavily penalizes any deviation from the target class. However, write depth d is an ordinal variable where adjacent bins represent similar retention strengths. To reflect this, the magnitude head is trained via Kullback-Leibler (KL) divergence against ordinal soft targets. If the expert deficit calculation targets depth d^*, the target distribution smears probability mass across adjacent available bins according to proximity, rather than placing a one-hot 1.0 on the nearest bin. This ensures that near-miss predictions (e.g., predicting $d = 80$ when the target is 110) are penalized less than catastrophic under-writes.

2. DAgger Expert Annealing During Phase 1, we use DAgger (Ross et al., 2011) to supervise the policy over the substrate. The probability β of executing the expert's target action versus the student policy's sampled action follows a linear decay schedule. For the first 5% of training (warmup), $\beta = 0.95$, ensuring the substrate remains sound while the trunk learns. Over the remainder of training, β decays linearly to 0.30, forcing the student to increasingly manage the consequences of its own imperfect writes.

Total Phase 1 Loss Formulation The complete multi-task objective during Behavioral Cloning is:

$$\begin{aligned}\mathcal{L}_{\text{total}} = w_{\text{ctrl}}\Big(&\lambda_g \mathcal{L}_{\text{BCE}}(\text{gate}) + \lambda_d \mathcal{L}_{\text{KL}}(\text{depth}) + \lambda_\tau \mathcal{L}_{\text{CE}}(\tau)\Big) \\ &+ \lambda_s \mathcal{L}_{\text{MSE}}(\text{stability})\end{aligned}$$

where w_{ctrl} is the warmup schedule scaler, and λ are static hyperparameter weights balancing the relative entropy of the action spaces.

B.6 Full Results: Additional Baselines

Table 12 gives the remaining systems omitted from Table 3 for space.

B.7 Expert Policy Details

The deficit-based expert (`capped_deficit`) computes, for each write event, a scalar deficit and returns an action tuple.

The foreign threshold F^* is calibrated per machine as the 75th percentile of F over training-set write events, producing roughly 25% of writes triggering non-trivial foreign discount.

System	CB	STABLE	ROUTINE	CONTESTED	TRANSIENT	ONESHOT
Naive	0.23	$0.14 \pm .02$	$0.50 \pm .04$	$0.29 \pm .04$	$0.11 \pm .02$	$0.12 \pm .02$
EWC	0.22	$0.16 \pm .02$	$0.50 \pm .04$	$0.29 \pm .04$	$0.12 \pm .02$	$0.09 \pm .02$
Oracle RAG (1.7B)	0.64	$0.45 \pm .03$	$0.58 \pm .03$	$0.63 \pm .04$	$0.86 \pm .03$	$0.71 \pm .03$
Oracle RAG (122B)	0.82	$0.84 \pm .02$	$0.75 \pm .03$	$0.75 \pm .03$	$0.79 \pm .03$	$0.99 \pm .01$

Table 12: Additional baselines omitted from Table 3 for space: weaker continual-learning variants and retrieval given ground-truth lifecycle metadata in place of a learned or LLM-managed store.

Algorithm 1 Deficit-based expert policy

Inputs: stability score s, `is_reinforcement` flag r, base decay b, self-steps on mask S, foreign-steps on mask F, depth bins $\{d_i\}$, foreign threshold F^*, tolerance ϵ, per-event cap $C = 150$.
Compute target retention depth $D^* = D^*(s, b, r)$ based on substrate curves.
Compute effective accumulated depth $D_{\text{acc}} = S - \gamma F$ with foreign discount $\gamma = \min(1, F/F^*)$.
Compute deficit $\Delta = D^* - D_{\text{acc}}$.
if $\Delta \leq \epsilon$ **then**
  return $(g = 0, d = 0, \tau = 0)$ {decline}
end if
Set write depth $d^* = \min(\Delta, C)$.
Snap d^* to nearest available bin.
Set $\tau^* = \arg\min_b |\log \tau_b - \log(24.5 \cdot b)|$.
return $(g = 1, d = d^*, \tau = \tau^*)$.

The target depth function $D^*(s, b, r)$ interpolates the substrate retrieval curve to determine what accumulated depth is needed for a fact of stability s and base decay b to remain retrievable through its expected query horizon. Reinforcement events use a slightly higher D^* than first-write events to preserve accumulation opportunity for future queries.

B.8 State Features

The controller conditions on six feature families summarized in Section 4.2. Full details:

Dense semantic embedding. A projection of the fact template's last-token representation from the base model, passed through an input normalization and a small MLP (`sem_proj`) that decorrelates raw LM features from lexical surface form. Dimension 256.

GRU hidden state. A recurrent summary of the trajectory so far, updated per event. Takes the current fact's dense embedding, event type indicator, and time delta since the previous event. Provides temporal context that a bag-of-features representation would lose. Dimension 256.

Rank ages. For each of the R rank columns, the time in days since that column was last written. Long-inactive columns are candidates for aggressive writes because their existing content has already decayed; recently touched columns carry current content that will be interfered with. Dimension R.

Slot norm. The L_2 norm of the mask-restricted LoRA weights, either measured directly on the substrate (during online training and evaluation) or predicted from ledger state via the fitted regression from Appendix A.2 (during offline training, for a $15\times$ speedup). Dimension 1.

Predicted stability. A scalar prediction from a dedicated head trained via MSE against the ground-truth stability score. Enables the controller to condition depth and gate decisions on predicted fact durability without needing symbolic access to metadata at inference.

Mask-level substrate features and forward-pass signal. For the specific mask that would receive this write, mean rank age within the mask, mean foreign-step count within the mask, and the forward pass test hit (Appendix A.2). The forward pass provides a substrate-derived signal of whether the incoming value is already represented in the current slot, complementing the rank-age and interference features.AUC against ground-truth reinforcement labels reaches 0.99. Dimension $N_{\text{rank feats}} + 1$.

B.9 TAU HEAD

The τ head is a five-way classifier over $\{25, 100, 400, 1600, 20000\}$ days. For a fact with base decay b, the target lifespan class is

$$\tau^*(b) = \arg\min_{\tau_j \in \mathcal{T}} |\log \tau_j - \log(24.5\, b)|\,,$$

where $\mathcal{T} = \{25, 100, 400, 1600, 20000\}$ days.

Class weighting. The τ head uses inverse-frequency class weights since the target distribution is more skewed. Bin 0 ($\tau = 25$) receives roughly 46% of events, bin 1 ($\tau = 100$) receives 10%, bin 3 ($\tau = 1600$) receives 2%. Inverse-frequency weighting maintains recall on rare bins at the cost of noisier training signal.

B.10 EMPIRICAL BIAS INITIALIZATION

Each action head's output bias is initialized to $\log(f)$ where f is the marginal frequency of each action in the expert policy on a calibration set of 150 users. Table 13 shows the initialized values.

Head	Initialized bias values
Store	0.99 (empirical write rate 72%)
Depth	$[-0.67, -4.47, -3.77, -2.01, -2.23, -2.03, -2.52]$ over $[20, 30, 40, 60, 80, 110, 150]$
τ	$[-0.77, -2.26, -1.35, -3.81, -1.88]$ over $[25, 100, 400, 1600, 20000]$

Table 13: Empirical bias initialization from expert marginals on 150-user calibration set.

The depth bias is heavily negative on rare bins (e.g., $d = 30$ at -4.47 reflects that only 1% of expert writes use this depth). The controller learns to adjust these biases per fact context and the initialization ensures early training does not have to first recover the marginal distribution.

B.11 LIFECYCLE-CLASS DEPTH PRIOR

The prior matrix $W \in \mathbb{R}^{5\times 7}$ maps predicted lifecycle-class probabilities to depth-preference adjustments.

Pattern	$d = 20$	$d = 30$	$d = 40$	$d = 60$	$d = 80$	$d = 110$	$d = 150$
STABLE	-0.46	-0.46	-0.46	-0.42	-0.19	$+0.99$	$+0.99$
ROUTINE	$+1.15$	$+0.99$	$+0.60$	-0.30	-0.74	-0.85	-0.85
CONTESTED	$+1.03$	$+1.03$	$+0.74$	-0.18	-0.75	-0.93	-0.93
TRANSIENT	$+1.15$	$+0.99$	$+0.60$	-0.30	-0.74	-0.85	-0.85
ONESHOT	-0.60	-0.31	$+0.11$	$+0.98$	$+0.98$	-0.31	-0.85

Table 14: Pattern-conditional depth prior matrix W. Each row is mean-centered so the prior does not shift aggregate write rate. The prior is combined with the pattern head's softmax as $\alpha_{\text{pattern}} \cdot p_{\text{pattern}} \cdot W$ and added to the depth head's raw logits.

A parallel stability-conditional depth prior encourages the depth head toward deep bins when predicted stability is high, using a seven-dimensional vector $[-1, -1, -0.5, 0, 0.5, 1.5, 1.5]$ over the depth bins, scaled by α_{stab} and the predicted stability scalar.

B.12 PER-ROUTER HYPERPARAMETERS

Building on the bias initialization (Appendix B.10) and depth priors (Appendix B.11) above, this section gives the remaining per-model hyperparameters. The prior scales differ between routers due to different substrate retrieval curves.

Hyperparameter	PMM-PR	PMM-LR
α_{pattern}	1.0	0.5
α_{stab}	2.5	0.5
w_τ base weight	1.0	1.0
w_τ^{boost}	4.0	4.0
τ boost duration (steps)	200	200
τ class weighting	inverse	inverse
Depth class weighting	sqrt	sqrt
DAgger β schedule	$0.95 \to 0.50$	$0.95 \to 0.50$
Controller warmup (steps)	100	100
Trunk learning rate	3×10^{-4}	3×10^{-4}
Controller learning rate	9×10^{-4}	9×10^{-4}
BC epochs	2	2
Batch size (users per step)	2	2

Table 15: Hyperparameters for the two model configurations. PMM-LR uses smaller prior scales because dynamic routing's permissive shallow-bin dynamics would otherwise push the controller to over-invest in deep writes.

B.13 STABILITY HEAD PRETRAINING

The stability head is pretrained via MSE regression against ground-truth stability scores over template-stability pairs from the training user pool. On 920 unseen templates (held-out from 34 users not seen during training), Stage A pretraining reaches $R^2 = 0.22$ at epoch 10, peaking slightly before plateauing.

The template-only prediction ceiling (best any model could achieve from the template embedding alone) is approximately 0.36. The gap from 0.22 to 0.36 is due to residual variance the head cannot extract from the embedding alone. Meanwhile, the gap from 0.36 to 1.00 represents the limit of this feature as identical fact templates have different true stabilities across different users depending on their personal patterns. This limit flows downstream to the τ head, which reads from the stability prediction.

B.14 STABLE ANALYSIS DETAILS

Per-query diagnostics on STABLE VAL outcomes were collected on 56 STABLE VAL queries across a 5-user analysis sample using the PMM-LR controller.

	Success ($n = 28$)	Failure ($n = 28$)
Prior writes (mean)	1.39	1.11
Last write depth (mean)	109.6	104.3
Accumulated depth (mean)	142.1	113.9
Days since last write (mean)	18.5	536.7
Self steps at query	124.1	57.2
Foreign steps at query	1127.8	1205.4
Foreign-to-self ratio (median)	8.6	23.2

Table 16: Per-query diagnostics on STABLE VAL. Successes and failures similar on writing behavior (depth, prior writes) but differ dramatically on query horizon and foreign-to-self ratio at query time.

The distinguishing features between success and failure are query horizon and foreign-to-self ratio at query time. Successful STABLE queries fire on average 18.5 days after last write; failed queries

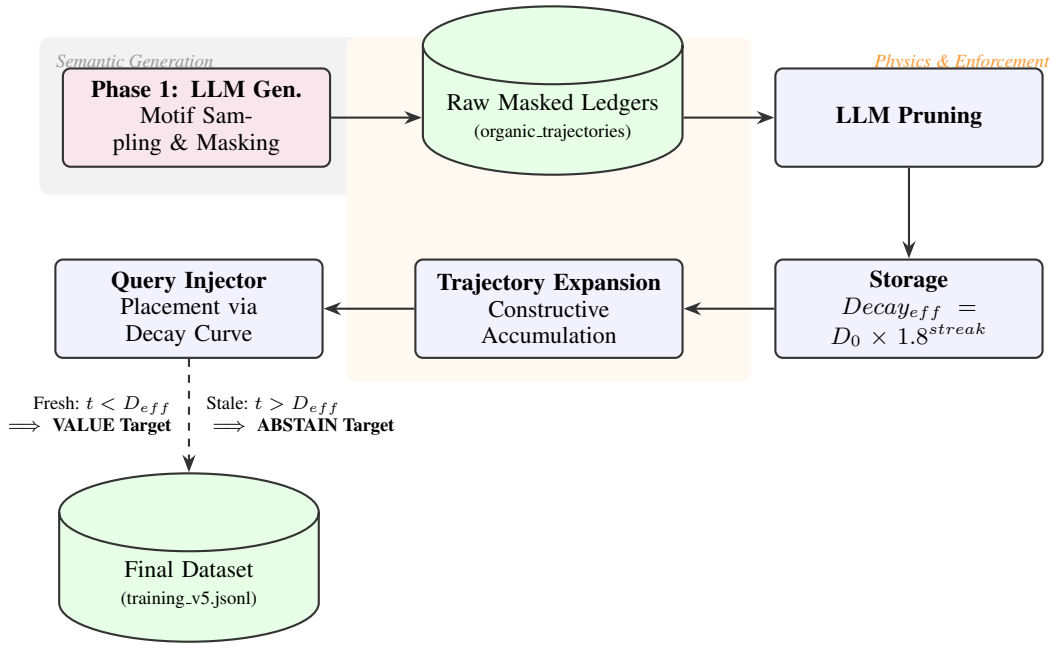

Figure 4: Overall dataset generation pipeline. Phase 1 grounds the data via LLM generation, while Phase 2 strictly enforces temporal dynamics, constructive accumulation, and delay-calibrated query injection.

fire 536.7 days after. Failed queries suffered from a $2.7\times$ higher foreign-to-self gradient ratio. This was not caused by encountering more absolute foreign traffic (means 1205 vs. 1128), but by the stable fact receiving far fewer reinforcing writes to defend its own trace against that traffic (57 vs. 124).

Max-τ ablation. An alternative τ update rule that preserves the longest τ any fact has assigned to a column (rather than overwriting) protects STABLE facts on shared columns from subsequent shorter-τ writes. This variant achieves class-balanced accuracy 0.45 (5-user eval), with STABLE VAL improving by $\sim$ 13 points but ABS accuracy collapsing across TRANSIENT, ROUTINE, CONTESTED, and ONESHOT (each below 0.05). TRANSIENT facts written to shared columns accidentally inherit the long τ of older STABLE facts. Consequently, these volatile facts never decay, causing the system to provide outdated answers instead of correctly abstaining during stale queries. This shows that the trade-off cannot be resolved by this simple max-τ update rule and reflects the coupling introduced by shared-column compression.

Reinforcement effect. STABLE facts written multiple times over their trajectory re-assert long τ on their columns and refresh self-investment against accumulated foreign traffic. STABLE facts with $3+$ writes hit at 0.68 versus 0.51 for single-write STABLE. The controller does not independently trigger reinforcement and it is driven by the upstream event stream identifying repeated observations of the same fact.

B.15 Phase 2 PPO Details

Phase 2 refines the phase 1 controller via PPO. Reward is $r = 2 \cdot \mathbb{1}[\text{correct}]$ for both VAL and ABS queries. The phase 1 checkpoint serves as the reference policy for KL regularization.

KL divergence to reference is penalized with weight 0.10 on the store head and 0.02 on depth. Batch size is 4 users per PPO iteration, with 3 PPO epochs and minibatch size 32. Value coefficient 0.5, auxiliary loss coefficient 0.3, entropy coefficient 0.015 on depth and 0.005 on the store head.

In our experiments, phase 2 provided marginal improvement over phase 1 (roughly ± 1 point class-balanced), primarily by shifting the policy's VAL-ABS balance without meaningfully improving

aggregate. We attribute this to phase 1 already being close to the metadata oracle ($\sim 86\%$), leaving little room for refinement via reward signal. We treat phase 2 as an optional stage.

B.16 Input Feature Ablations

To understand which information is strictly necessary for parametric memory management, we performed a set of input feature ablations. Because the controller uses a shared representation trunk that feeds multiple heads, simply removing an architectural head could confound the results. Instead, we ablate by zeroing out the specific input sources entirely before they reach the controller, directly testing the information bottleneck. We evaluate four ablations against the PMM-PR baseline:

1. **No Semantic:** Zeroes out the dense text embedding.
2. **No Priors:** Removes the Stage A stability warm-start and the mathematical depth-bias.
3. **No History:** Zeroes out the GRU trajectory hidden state.
4. **No Substrate:** Zeroes out substrate readings (`probe_hit`, `slot_norm`, `foreign_steps`).

		STABLE		ROUTINE		CONTESTED		TRANSIENT		ONESHOT	
System	CB	VAL	ABS	VAL	ABS	VAL	ABS	VAL	ABS	VAL	ABS
PMM-PR	**0.60**	0.33	0.81	0.80	0.64	0.81	0.34	0.66	0.58	0.82	0.58
No History	0.58	0.33	0.91	0.67	0.68	0.67	0.32	0.74	0.68	0.58	0.76
No Substrate	0.58	0.34	0.95	0.77	0.55	0.76	0.28	0.70	0.55	0.65	0.72
No Priors	0.52	0.42	0.80	0.66	0.57	0.73	0.20	0.78	0.41	0.69	0.44
No Semantic	0.47	0.28	0.88	0.70	0.33	0.63	0.28	0.61	0.35	0.49	0.60

Table 17: Input feature ablations. Losing semantic features or priors causes substantial failure. History and Substrate act as a dual-redundancy system, but isolating them reveals specific, severe capability collapses (e.g., losing history crashes ONESHOT VAL by 24 points).

The per-class VAL/ABS splits (Table 17) reveal exactly how the controller uses its inputs, surfacing three core findings:

1. Semantic embeddings and priors provide substantial management signal. Without semantic embeddings, the model cannot map an incoming fact to a behavioral pattern. Without this information, the policy falls back toward less differentiated write behavior, reducing its ability to separate persistent and volatile events. Removing semantic features lowers CB to 0.47, while removing the substrate-derived priors lowers it to 0.52.

2. History and substrate state provide complementary information. The *No History* (CB 0.58) and *No Substrate* (CB 0.58) ablations survive in aggregate because the GRU and the LoRA state provide overlapping signals for reinforcement. If the GRU is removed, the substrate's `probe_hit` still tells the controller a fact is recurring. If the substrate is removed, the GRU's temporal memory knows it has seen the embedding multiple times. However, the class-wise splits show they are not perfectly redundant:

- **History disambiguates lifecycle from trajectory context:** Without the GRU, the model is blind to the timeline. When a ONESHOT fact arrives, the model cannot confidently assign it moderate depth because it lacks the behavioral context to separate it from the first write of a ROUTINE fact. Consequently, it hedges its bets, and ONESHOT VAL crashes by 24 points ($0.82 \rightarrow 0.58$).
- **Substrate state exposes the consequences of prior writes:** Without the substrate, the model cannot see collision metrics (`foreign_steps`, `slot_norm`). Blind to overlap, it overwrites frequently. This leaves stale traces persisting when they should have decayed, dropping abstention (ABS) accuracy on volatile, highly-reinforced classes (ROUTINE ABS drops $0.64 \rightarrow 0.55$; CONTESTED ABS drops $0.34 \rightarrow 0.28$).

3. Higher abstention under ablation reflects more conservative writing. STABLE and ONESHOT ABS actually improve under these ablations (climbing to 0.91-0.95), but this is not a genuine gain as the controller compensates for information loss by defaulting to shallower writes, which decay faster and inflate ABS at the cost of VAL.

C Dataset and Trajectory Generation Details

This appendix provides extended statistics, motif distributions, and qualitative examples for the controlled long-horizon user trajectories introduced in Section 2.

C.1 Dataset Statistics

The dataset consists of 909 user trajectories spanning multi-year horizons. Table 18 summarizes the aggregate dataset statistics and the class balance across the five memory classes. Figure 5 visualizes how identical fact properties map to different classes, while Figure 6 illustrates the query injection density across a fact's decay threshold.

Metric	**Value**
Total Users	909
Query Horizon	Avg 5, 434 days (Min: 309, Max: 25, 282)
Total Writes	170, 121
Total Queries	144, 164
Query Target Split	69.3% VALUE | 30.7% ABSTAIN

Classes	**% of Writes**	**% of Queries**
STABLE	27.2%	46.7%
ROUTINE	41.0%	24.3%
CONTESTED	20.5%	11.4%
TRANSIENT	7.7%	8.9%
ONESHOT	3.7%	8.6%

Table 18: At-a-glance dataset statistics (top) and class balance distributions (bottom). Queries are intentionally reweighted to ensure critical behaviors like expiration (TRANSIENT) are adequately represented despite write imbalances. Query horizon is separate from write events horizon of 700-1500 days.

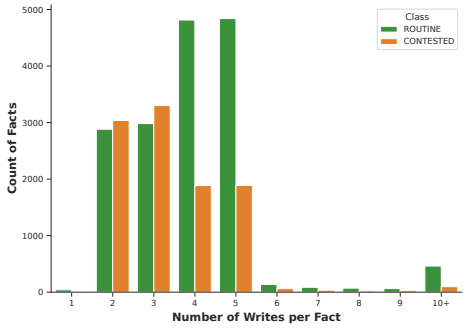

Figure 5: Two facts can have very similar properties such as stability and number of writes, yet belong to very different classes based on user history and semantic meaning.

C.2 Motif Distribution and Cross-User Variance

To ensure heterogeneous, user-level memory dynamics, user personas were generated using a variety of underlying temporal motifs governing their behavior (Table 19). Because these motifs dictate behavior at the user level, the same fact type (e.g., a hobby or a job) can be stable for one user but volatile for another. Figure 7 illustrates the resulting cross-user variance in routine predictability.

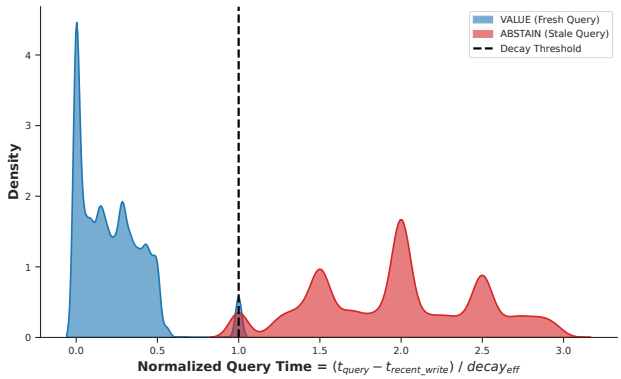

Figure 6: Fresh versus stale query injection density. Retrieval (VAL) and abstention (ABS) queries are placed relative to each fact's validity horizon.

Temporal Motif	Users
NORMAL DRIFT: Slow, mundane, almost imperceptible changes	347
THE MICRO-VARIATION: Highly stable daily routine	75
THE SEASONAL SHIFT: 2-to-3 month block behavior	67
THE PHASED TRANSITION: Gradual shift over 2 months	59
THE VACATION/TRIP: Stable routine changes for 2 weeks	51
THE INSTANT CORRECTION: Realizes mistake minutes/hours later	45
THE CASCADE EVENT: Single major event triggers 3-4 routines	41
THE FAILED HABIT: Tries for 5 days, then drops completely	36
THE CONTEXT-DEPENDENT TOGGLE: Toggles based on day of week	33
THE RELAPSE: Abandoned habit returns heavily for 2 weeks	30
DORMANT RESURGENCE: Ignored for 6 months, then relevant	26
THE RAPID PROMOTER: Switches roles every 3-4 months	24
THE VOLATILE ROUTINE: Constantly breaks habit and gets back	22
THE NOMAD: Moves and changes routine every 6 months	20
THE CHAMELEON: Adopts completely new hobbies every season	18
THE BINGE: Intense, short-term burst never happening again	15

Table 19: Distribution of the underlying temporal motifs used to govern user memory dynamics during dataset generation.

C.3 Qualitative Example

Table 20 provides a qualitative snippet of a user trajectory demonstrating how writes, updates, and queries are interleaved. Entities are masked to decouple semantic world knowledge from parametric retention.

C.4 Benchmark Comparison Criteria

Table 1 compares prior long-term memory benchmarks along the following properties:

Belief revision. Whether later observations can supersede an earlier value and evaluation requires recovering the currently valid state.

Time-based expiry. Whether previously valid information can become invalid solely because its validity horizon has elapsed, without an explicit replacement, correction, or deletion event.

Reinforcement. Whether repeated consistent observations affect subsequent memory behavior, such as strengthening retention, rather than being treated as independent events.

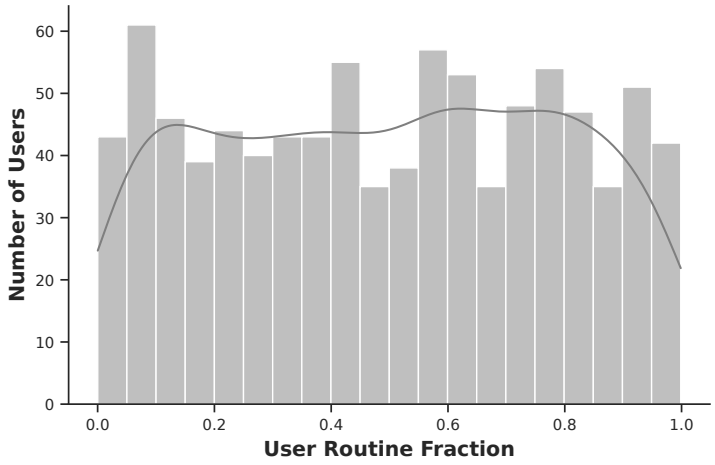

Figure 7: Cross-user variance in routine prevalence. The variation across users illustrates why lifespans cannot be determined solely from fact type and must also depend on individual trajectory context.

Day	Event	Pattern	Ledger Entry / Query
Day 0	INTRODUCE	STABLE	I work as a [JOB_8]
Day 2	INTRODUCE	ROUTINE	I ordered my morning coffee from [LOC_CAFE_4]
Day 5	INTRODUCE	TRANSIENT	I grabbed a quick lunch with Sarah at [LOC_BISTRO_9]
Day 12	ABS QUERY	TRANSIENT	I grabbed a quick lunch with Sarah at [MASK]
Day 14	INTRODUCE	CONTESTED	My current priority project is [PROJ_2]
Day 15	UPDATE	ROUTINE	I ordered my morning coffee from [LOC_CAFE_4]
Day 18	VAL QUERY	ROUTINE	I ordered my morning coffee from [MASK]
Day 22	CONTRADICT	CONTESTED	My current priority project is [PROJ_7]
Day 25	VAL QUERY	CONTESTED	My current priority project is [MASK]
Day 30	INTRODUCE	ONESHOT	I booked a 2-week vacation to [LOC_1]
Day 45	VAL QUERY	ONESHOT	I booked a 2-week vacation to [MASK]
Day 400	VAL QUERY	STABLE	I work as a [MASK]

Table 20: A narrative snippet of a user trajectory demonstrating all five memory lifecycles. The example illustrates the distinct target behaviors: TRANSIENT information is queried after its validity horizon, CONTESTED information is queried after a value revision, and STABLE information must remain retrievable across long intervening histories. The non-query inputs also follow the same [MASK]-Output format to memorize as the query inputs but are presented in a readable form here with outputs values inserted in their respective [MASK] positions.

User-conditioned lifespan. Whether the appropriate lifespan of a memory depends on the individual user's observed history rather than only on the semantic type of the information.

Lifecycle supervision. Whether the benchmark provides event-level ground truth describing how a memory's validity or state evolves over time.

We mark a property as *full* when it is explicitly constructed and evaluated, *partial* when related behavior is present but not directly isolated, and *absent* otherwise.

C.5 Dataset Schema and Comprehensive Trajectory Example

Because the full dataset consists of multi-year trajectories with hundreds of events per user, we provide a detailed example here to illustrate the exact inputs, metadata, and query formats utilized by both our parametric controllers and the retrieval-based baselines.

Each user in the dataset is represented as a JSON object containing a `facts_metadata` dictionary (the ground-truth for each fact) and an `events` array (the chronological stream of writes and queries).

1. Ground-Truth Metadata Dictionary The `facts_metadata` dictionary maps each semantic fact type (e.g., *favorite_coffee_shop*) to its ground-truth lifecycle properties which represents a user's belief states. This metadata is strictly hidden from the controller at inference time but is used by the DAgger oracle during Phase 1 training to compute target depths.


```
"facts_metadata": {
  "user_profession": {
    "write_pattern": "STABLE",
    "stability_score": 0.92,
    "base_decay_days": 3650.0
  },
  "morning_coffee_spot": {
    "write_pattern": "ROUTINE",
    "stability_score": 0.45,
    "base_decay_days": 14.0
  },
  "current_project": {
    "write_pattern": "CONTESTED",
    "stability_score": 0.65,
    "base_decay_days": 90.0
  },
  "todays_lunch": {
    "write_pattern": "TRANSIENT",
    "stability_score": 0.12,
    "base_decay_days": 2.0
  }
}
```


2. Chronological Event Stream The `events` array represents the timeline. Each event contains a timestamp, the event type (`INTRODUCE`, `UPDATE`, `CONTRADICT`, or `QUERY`), and the event.

Below is an illustrative sequence showing how the `morning_coffee_spot` (a ROUTINE fact) and `todays_lunch` (a TRANSIENT fact) are processed over a 20-day window.

Event 1: First observation of a routine (Day 0)


```
{
  "clock_days": 0.0,
  "event_type": "INTRODUCE",
  "fact_type": "morning_coffee_spot",
  "value": "CAFE_4",
  "is_reinforcement": false,
  "template_input": "I ordered my morning coffee from [MASK]",
  "natural_language": "[Clock: 0000] I ordered my morning coffee
  from CAFE_4."
}
```


Event 2: A transient, unrepeated event (Day 2)


```
{
  "clock_days": 2.5,
  "event_type": "INTRODUCE",
  "fact_type": "todays_lunch",
  "value": "RESTAURANT_9",
  "is_reinforcement": false,
```

```
  "template_input": "I grabbed a quick lunch with Sarah at [MASK]",
  "natural_language": "[Clock: 0002] I grabbed a quick lunch with
  Sarah at RESTAURANT_9."
}
```

Event 3: Reinforcement of the routine (Day 5)

```
{
  "clock_days": 5.0,
  "event_type": "UPDATE",
  "fact_type": "morning_coffee_spot",
  "value": "CAFE_4",
  "is_reinforcement": true,
  "template_input": "I ordered my morning coffee from [MASK]",
  "natural_language": "[Clock: 0005] I ordered my morning coffee
  from CAFE_4."
}
```

Event 4: Stale Abstention Query for the Transient fact (Day 15)
Because `todays_lunch` has a base decay of 2.0 days, and 12.5 days have passed since Event 2, the ground-truth target strictly demands abstention.

```
{
  "clock_days": 15.0,
  "event_type": "QUERY",
  "target_type": "ABSTAIN",
  "fact_type": "todays_lunch",
  "target_value": "Unknown",
  "template_input": "I grabbed a quick lunch with Sarah at [MASK]",
  "natural_language": "Where did you grab lunch with Sarah?"
}
```

Event 5: Value Retrieval Query for the Routine fact (Day 18)
Because the routine was reinforced at Day 5, the controller should have accumulated depth, keeping the memory valid for this query 13 days later.

```
{
  "clock_days": 18.0,
  "event_type": "QUERY",
  "target_type": "VALUE",
  "fact_type": "morning_coffee_spot",
  "target_value": "CAFE_4",
  "template_input": "I ordered my morning coffee from [MASK]",
  "natural_language": "Question: Where do you usually
  order your morning coffee?"
}
```

D NATURAL-LANGUAGE INTERFACING AS A SEPARABLE CAPABILITY

The main paper isolates memory-management dynamics using a templated interface (§3). This appendix pairs the same substrate with a natural-language (NL) retrieval adapter and characterizes its behavior, establishing that NL interfacing is a separately trainable capability whose current gap to templated retrieval is expected and does not reflect a limitation of the substrate or the controller.

Setup. We train two retrieval adapters on the same memory substrate. The *template* adapter consumes templated queries in the write format (e.g., *I took my dog to [MASK]*). The *NL* adapter consumes free-form questions targeting the same underlying facts (e.g., *Where did I take my dog?*). Both adapters use the rank-32 architecture and depth-conditioned training procedure of §3.3, applied to identical substrate states. The NL adapter is trained with a modest number of question paraphrases

(3-4) per fact template and is deliberately not optimized further. The purpose of this study is to establish trainability and separability, not to match templated performance.

Training behavior. The template adapter reaches near-saturation within the first few hundred training steps. Retrieval accuracy and abstention accuracy both remain high throughout, with deep writes recovered reliably and empty slots correctly marked `Unknown` from the outset. The NL adapter begins near zero on both metrics and improves gradually. This gap reflects the difference in task scope. The template adapter calibrates only a depth-to-value/unknown mapping under a fixed pattern, while the NL adapter must additionally learn a mapping from open-ended question forms to the same underlying slot state.

Adapter	Recipe	n	`gen==V`	`gen==U`
Template	single_deep	125	99.2%	0.0%
Template	multi	75	100.0%	0.0%
Template	single_shallow_med	100	100.0%	0.0%
Template	empty	100	0.0%	100.0%
NL	deep	336	71.1%	17.3%
NL	medium	171	46.2%	36.8%
NL	multi	335	61.8%	30.1%
NL	shallow_single	163	10.4%	72.4%
NL	empty	329	0.0%	88.1%

Table 21: Retrieval adapter behavior on held-out queries. `gen==V` is the fraction of queries for which the adapter emits the correct value. `gen==U` is the fraction for which it emits `Unknown`.

Held-out retrieval behavior. Table 21 reports final adapter behavior by write depth. The NL adapter retrieves reliably on deep and multi-write recipes (61–71% correct value) and abstains reliably on empty slots (88.1%), but underperforms on medium and shallow writes. Its dominant failure mode is conservative abstention rather than hallucination. On shallow single writes, the adapter emits `Unknown` 72.4% of the time when a value should be retrieved.

End-to-end evaluation. Table 22 reports the causal oracle controller (§4) paired with the NL retrieval adapter and evaluated with NL queries on held-out users. Class-balanced accuracy is 0.444. Per-pattern behavior is consistent with the retrieval adapter's conservative bias as patterns dominated by abstention queries (TRANSIENT, 85% ABS) benefit from the adapter's willingness to withhold values under weak evidence, while patterns with a larger retrieval share are penalized. Substrate write, decay, and reinforcement dynamics are identical to the templated evaluation, so the gap is attributable to the retrieval adapter alone.

Pattern	Accuracy	VAL_n	VAL	ABS_n	ABS
STABLE	0.414	1029	0.393	169	0.544
ROUTINE	0.452	541	0.412	245	0.539
CONTESTED	0.351	286	0.413	338	0.299
TRANSIENT	0.591	104	0.250	608	0.650
ONESHOT	0.415	604	0.561	660	0.280
Class-balanced	**0.444**				

Table 22: Metadata oracle controller paired with the natural-language retrieval adapter, evaluated with natural-language queries.

Interpretation. These results support two claims. First, natural-language retrieval on the proposed substrate is trainable. The same retrieval architecture that saturates on templated queries reaches non-trivial retrieval and reliable empty-slot abstention on free-form queries with modest training. Second, natural-language retrieval is separable from lifecycle management. The retrieval adapter operates over slot state, so the substrate can be paired with either interface without altering its dynamics or its policy.

The gap between the template and NL adapters is expected and independently characterized in prior work. Allen-Zhu & Li (2024) show that reliable knowledge extraction from parametric memory requires paraphrase augmentation during training, without which memorization does not translate into extractable knowledge under novel question wordings. In the fine-tuning regime that matches ours, Mecklenburg et al. (2024) report that question-answering capability scales with the number of paraphrases per fact and saturates near ten paraphrases per sample. Our proof-of-concept adapter operates well below this regime. Optimizing the natural-language interface following standard paraphrase-augmentation practices is outside the scope of this work, which focuses on the substrate and the lifecycle policy.