RPMEM: LEARNING LONG-TERM RECURRENT PARAMETRIC MEMORY ACROSS SESSIONS FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-running LLM agents require memory that persists and evolves across sessions. Text-based memory retrieves and reconstructs past interactions at every query, making long-horizon performance increasingly dependent on retrieval quality and contextual reasoning as histories grow. Parametric memory encodes experience directly into model computation, but existing approaches provide limited support for cross-session memory evolution. Their coupling to a specific backbone further restricts memory reuse after model replacement. We introduce RPMem, a two-stage architecture that compiles each session into a model-independent latent memory through forward computation and selectively integrates it with retained memory via a task-trained recurrent gate. The consolidated memory is then mapped to backbone-specific low-rank adaptation (LoRA) parameters, allowing the encoding capability to transfer when the backbone is replaced. Evaluation across three long-term memory benchmarks and five diverse backbones demonstrates broad generalization with near-constant update cost and memory footprint. With Qwen3-8B on PERMA, RPMem reaches 85.52%, outperforming the strongest parametric and text-based baselines by 5.32 and 12.98 percentage points, respectively. Ablations validate the complementary roles of session compilation and cross-session consolidation, while dynamics analyses reveal that the gate acquires task-specific memory integration strategies. These results establish RPMem as a lifecycle-independent parametric memory framework that maintains evolving cross-session memory that remains reusable across backbone replacements. Our implementation is available in an anonymous repository at code.

1 INTRODUCTION

LLM agents increasingly operate across multiple sessions, requiring continuity in both interaction and task execution. Sustained user interaction depends on retaining relevant historical context (Cao et al., 2026; Zhao et al., 2025; Jiang et al., 2025). Task execution across episodes similarly requires preserving prior decisions, constraints, and outcomes (Park et al., 2023; Packer et al., 2023; Zhong et al., 2024). Long-term memory has therefore become an essential capability for agent architectures (Hu et al., 2025).

The dominant approach to long-term memory stores past interactions as external text and retrieves relevant information at query time. Recent work has refined this approach by extracting salient information and consolidating it into compact, retrievable records (Chhikara et al., 2025; Fang et al., 2026). Adaptive memory organization further links related entries and revises existing records as new information arrives (Xu et al., 2025). Reinforcement learning has also been introduced to optimize memory operations and the selection of retrieved information for downstream tasks (Yan et al., 2026). This text-based formulation keeps stored memories inspectable and separate from the backbone. Despite improvements in organization and management, using these memories still requires retrieving relevant records and integrating their contents into the input context at each query. When evidence is distributed across sessions, incomplete retrieval may omit information necessary for reasoning. Expanding the retrieved context can improve coverage, but does not ensure that the model effectively uses the available evidence (Liu et al., 2024).

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

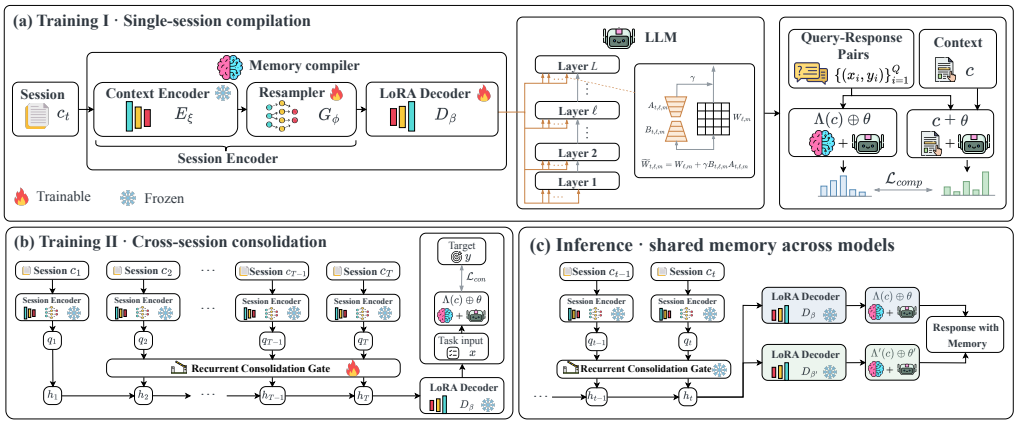


Figure 1: Overview of the RPMem architecture. (a) Single-session compilation trains the Perceiver-based resampler and LoRA decoder to match context-conditioned responses, with the context encoder and backbone frozen. (b) Cross-session consolidation trains only the recurrent gate to integrate session memories under task supervision. (c) At inference, accumulated memory is updated recurrently and decoded into backbone-specific LoRA parameters. Adapted decoders enable reuse of the same memory across backbones; all modules are frozen.

Parametric memory offers an alternative by encoding experience directly into model computation, allowing memory to inform future reasoning without occupying the input context (Charakorn et al., 2026; Liu et al., 2026c; Tang et al., 2026; Zhang et al., 2026). However, deploying parametric memory in long-lived agents requires more than encoding a single context into parameters. Memory updates through forward computation are desirable for scalable deployment, as they avoid the overhead of repeated gradient-based optimization. Methods such as Doc-to-LoRA and SHINE achieve this by mapping bounded text to low-rank adaptation (LoRA) parameters (Hu et al., 2022) in a single pass (Charakorn et al., 2026; Liu et al., 2026c), but each session produces a static snapshot without a mechanism for incremental cross-session updates.

Selectively retaining, revising, and integrating information across sessions raises a more challenging requirement. MEMORYLLM and M+ maintain updatable memory within the language model (Wang et al., 2024c; 2025a), while Metis learns forward-updated memory matrices within a memory-augmented backbone (Zhang et al., 2026). Yet in existing parametric approaches, memory representations are defined by and coupled to a specific backbone architecture. In deployed systems, the serving model is routinely replaced, and under this coupling accumulated memory cannot be directly reused. Existing adapter-transfer methods support reuse of learned task adaptations across model changes (Wang et al., 2024b; Gu et al., 2025; Jung & Kim, 2026). Persistent agent memory additionally requires continuity of cross-session updates through those changes. This lifecycle dependence is, we argue, the central unsolved barrier to persistent parametric memory. We define lifecycle independence as preserving and continuing to use accumulated memory across serving-backbone replacements.

We introduce RPMem (Recurrent Parametric Memory), an architecture that separates persistent memory from the serving backbone, as illustrated in Figure 1. Training proceeds in two stages. Single-session compilation learns to encode context into a shared latent memory space and decode it into backbone-specific LoRA parameters. Cross-session consolidation trains a lightweight recurrent gate under task supervision to integrate session memories into a fixed-size state. At inference, this state is updated through forward computation and decoded into LoRA parameters for the frozen backbone. Backbone replacement requires adapting only the decoder, preserving the accumulated memory and recurrent updater so that memory remains usable and continues evolving. Our contributions are summarized below:

1. We formulate the problem of lifecycle-independent parametric memory and identify the requirements it imposes on memory writing, cross-session evolution, and backbone decoupling.
2. We propose RPMem, to our knowledge the first lifecycle-independent parametric memory architecture, which maintains evolving cross-session memory in a model-independent latent space that is decoded into LoRA parameters for the current frozen backbone.

3. Experiments across three benchmarks and five backbones, together with ablation and memory-dynamics analyses, validate the two-stage design and demonstrate that parametric memory can be decoupled from the backbone lifecycle.

2 METHOD

2.1 OVERVIEW

We consider an agent that receives an ordered stream of bounded interaction sessions $\mathcal{C}_{1:T} = (c_1, \ldots, c_T)$ and later answers a query x. Here, T is the number of observed sessions, $t \in \{1, \ldots, T\}$ indexes their arrival order, and c_t denotes the t-th session. A frozen autoregressive language model f_θ, whose backbone parameters are denoted by θ, generates a response y according to

$$p(y \mid x, \mathcal{C}_{1:T}) = \prod_{j=1}^{N} p(y_j \mid x, y_{<j}, \mathcal{C}_{1:T}), \tag{1}$$

where $N = |y|$ is the response length, $j \in \{1, \ldots, N\}$ is the generation position, y_j is the token generated at position j, and $y_{<j}$ is its preceding response prefix. Our goal is a fixed-size persistent memory that carries information from $\mathcal{C}_{1:T}$ into generation and incorporates each new session without reprocessing earlier ones.

RPMem organizes memory processing into session encoding, recurrent consolidation, and model-specific decoding. Single-Session Memory Compilation (Figure 1(a)) learns to encode each session into fixed-shape latent memory and decode it into LoRA parameters that preserve the session's influence on the frozen backbone's responses. Cross-Session Memory Consolidation (Figure 1(b)) then uses downstream supervision to learn how to selectively integrate successive session memories through a recurrent gate. The model-specific decoder maps the consolidated memory to LoRA parameters for response generation.

2.2 SINGLE-SESSION MEMORY COMPILATION

The memory compiler is a hypernetwork comprising a session encoder and a model-specific LoRA decoder. The session encoder combines a frozen context encoder with a trainable mapping that compresses variable-length context features into fixed-shape session memory. For a tokenized session c_t, these operations are

$$X_t = E_\xi(c_t) \in \mathbb{R}^{L \times n_t \times d_e}, \tag{2}$$

$$q_t = G_\phi(X_t) \in \mathbb{R}^{L \times M \times r \times d}. \tag{3}$$

Here, the frozen context encoder E_ξ, with parameters ξ and hidden width d_e, extracts features for the n_t tokens of c_t from L evenly spaced depths, one per adapted layer of the compilation backbone. The axes of X_t index encoder depth, token position, and feature. We implement G_ϕ, with trainable parameters ϕ, as a Perceiver-based resampler (Jaegle et al., 2021) that independently compresses each layer's token features using shared parameters. In its output q_t, M counts the types of linear modules receiving LoRA updates within each adapted layer. The symbols r and d denote the LoRA rank and latent feature dimension, respectively. Each slice $q_{t,\ell,m,k} \in \mathbb{R}^d$ represents rank component $k \in \{1, \ldots, r\}$ for module type $m \in \{1, \ldots, M\}$ at layer $\ell \in \{1, \ldots, L\}$. The overall shape of q_t is therefore independent of the session length n_t.

A decoder maps q_t to a model-specific adapter collection and applies the resulting LoRA updates to the corresponding frozen backbone weights:

$$D_\beta(q_t) = \Lambda_t = \{A_{t,\ell,m}, B_{t,\ell,m}\}_{\ell,m}, \tag{4}$$

$$\widetilde{W}_{t,\ell,m} = W_{\ell,m} + \gamma B_{t,\ell,m} A_{t,\ell,m}. \tag{5}$$

Here, D_β denotes the decoder with trainable parameters β, and Λ_t collects the generated factor pairs across layers ℓ and module types m. $A_{t,\ell,m} \in \mathbb{R}^{r \times d^{\text{in}}_{\ell,m}}$ is the down-projection factor and $B_{t,\ell,m} \in \mathbb{R}^{d^{\text{out}}_{\ell,m} \times r}$ is the up-projection factor generated from session c_t. The symbols $d^{\text{in}}_{\ell,m}$ and $d^{\text{out}}_{\ell,m}$ denote the input and output widths of the original linear map $W_{\ell,m} \in \mathbb{R}^{d^{\text{out}}_{\ell,m} \times d^{\text{in}}_{\ell,m}}$. The fixed

coefficient γ scales the LoRA update, and $\widetilde{W}_{t,\ell,m}$ denotes the resulting session-adapted map. We write $\theta \oplus \Lambda_t$ for the frozen backbone augmented with the generated adapter collection.

We train the memory compiler to encode session memories in latent space and make them available through generated LoRA parameters. For each session, we construct query–response pairs. Given the same query and reference response prefix, we align the next-token distribution of the frozen backbone equipped with the generated LoRA with that of the same backbone reading the session context without an adapter:

$$p^{\mathrm{ctx}}_{i,j}(v) = p_\theta(v \mid c, x_i, y_{i,<j}), \tag{6}$$

$$p^{\mathrm{mem}}_{i,j}(v) = p_{\theta \oplus \Lambda(c)}(v \mid x_i, y_{i,<j}). \tag{7}$$

Here, each session c has Q query–response pairs $\{(x_i, y_i)\}_{i=1}^{Q}$, indexed by i, with a query x_i and a fixed reference response $y_i = (y_{i,1}, \ldots, y_{i,N_i})$ of length N_i. At response position $j \in \{1, \ldots, N_i\}$, $y_{i,j}$ denotes the token and $y_{i,<j} = (y_{i,1}, \ldots, y_{i,j-1})$ its preceding prefix. The collection $\Lambda(c) = D_\beta(G_\phi(E_\xi(c)))$ is the adapter generated from session c. The symbols $p^{\mathrm{ctx}}_{i,j}$ and $p^{\mathrm{mem}}_{i,j}$ denote the context-conditioned and adapter-conditioned next-token distributions, respectively, while v indexes a candidate token in their shared output vocabulary. Construction and split details are provided in Appendix C.

We retain the K highest-probability tokens under the context-conditioned distribution. Both distributions use this same token set and aggregate the remaining probability mass into a single tail category. The selected set, tail mass, and token-level forward-KL loss are

$$S_{i,j} = \mathrm{TopK}_v\big(p^{\mathrm{ctx}}_{i,j}(v), K\big), \qquad \tau^a_{i,j} = 1 - \sum_{v \in S_{i,j}} p^a_{i,j}(v), \tag{8}$$

$$\ell_{i,j} = \sum_{v \in S_{i,j}} p^{\mathrm{ctx}}_{i,j}(v) \log \frac{p^{\mathrm{ctx}}_{i,j}(v)}{p^{\mathrm{mem}}_{i,j}(v)} + \tau^{\mathrm{ctx}}_{i,j} \log \frac{\tau^{\mathrm{ctx}}_{i,j}}{\tau^{\mathrm{mem}}_{i,j}}. \tag{9}$$

Here, $\mathrm{TopK}_v(\cdot, K)$ returns the set of K token indices with the highest probabilities, so $|S_{i,j}| = K$. The superscript $a \in \{\mathrm{ctx}, \mathrm{mem}\}$ identifies one of the two distributions, $\tau^a_{i,j}$ is its probability mass outside $S_{i,j}$, and $\ell_{i,j}$ is the forward-KL loss at position j of reference response y_i. The compiler averages $\ell_{i,j}$ over positions within each response and then over query–response pairs to obtain the session loss $\mathcal{L}_{\mathrm{FKL}}(c)$ and the regularized compilation objective $\mathcal{L}_{\mathrm{comp}}$:

$$\mathcal{L}_{\mathrm{FKL}}(c) = \frac{1}{Q} \sum_{i=1}^{Q} \frac{1}{N_i} \sum_{j=1}^{N_i} \ell_{i,j}, \tag{10}$$

$$\mathcal{L}_{\mathrm{comp}} = \mathbb{E}_c[\mathcal{L}_{\mathrm{FKL}}(c) + \lambda \mathcal{R}_1(c)]\,. \tag{11}$$

Here, $\mathbb{E}_c$ averages over sessions in the compilation corpus, $\mathcal{R}_1(c)$ averages the absolute magnitudes of all LoRA factors decoded from c, and λ is the regularization weight. Optimization updates ϕ and β while holding ξ and θ fixed.

The separation between G_ϕ and D_β supports adaptation to a target backbone $f_{\theta'}$, whose frozen parameters are denoted by θ'. We retain G_ϕ, instantiate a target-specific decoder $D_{\beta'}$ with parameters β', and optimize β' with the same fixed-reference forward-KL objective while the memory encoder and target backbone remain fixed. The target decoder accepts the unchanged memory representation and generates LoRA factors matching the target backbone's layer count and module dimensions.

2.3 Cross-Session Memory Consolidation

Compilation places every q_t in the same fixed-shape memory space. Let $h_t \in \mathbb{R}^{L \times M \times r \times d}$ denote the accumulated memory after the first t sessions. We set $h_0 = 0$ before any session arrives and initialize the memory directly from the first session as $h_1 = q_1$. For each subsequent session, a lightweight recurrent consolidation gate combines the accumulated memory with the incoming representation through a retention tensor:

$$z_t = \sigma([h_{t-1}; q_t] W_g + b_g)\,, \qquad t \geq 2. \tag{12}$$

The tensor $z_t \in \mathbb{R}^{L \times M \times r \times d}$ contains a d-dimensional retention vector at every layer, module, and rank index. The operator $[\,;\,]$ denotes concatenation, $W_g \in \mathbb{R}^{2d \times d}$ is the shared gate projection, $b_g \in$

$\mathbb{R}^d$ is its bias, and σ is the element-wise sigmoid. Concatenation and the shared linear projection operate along the latent feature dimension. The gate updates the persistent state as

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot q_t, \qquad t \geq 2, \tag{13}$$

where $\odot$ denotes element-wise multiplication and 1 is an all-ones tensor with the same shape as z_t. Thus, z_t weights the previous state and $1-z_t$ weights the incoming session representation. The same gate is shared across every layer, module, and rank index, yielding $2d^2 + d$ parameters independent of L, M, and r.

Compilation learns to encode individual sessions as parametric memory through distribution matching. We separate consolidation learning from memory compilation to tailor memory retention and updating to the information needs and interaction patterns of the target application scenario. The second stage trains the gate with supervision from that scenario, and the learned policy is reused across subsequent interactions after deployment. For a training example $(\mathcal{C}_{1:T}, x, y)$, the compiler produces the ordered representations $q_1, \ldots, q_T$, the recurrence forms the final persistent state h_T, and the decoder produces the corresponding adapter collection $\Lambda_T = D_\beta(h_T)$. We collect the trainable gate parameters as $\psi = \{W_g, b_g\}$ and optimize the expected loss over the target application's training distribution:

$$\mathcal{L}_{\text{con}}(\psi) = \mathbb{E}_{(\mathcal{C}_{1:T}, x, y) \sim \mathcal{D}_{\text{task}}} \left[\ell_{\text{task}}(p_{\theta \oplus \Lambda_T}(\cdot \mid x), y) \right], \tag{14}$$

where $\mathcal{D}_{\text{task}}$ is the training-data distribution of the target application, $p_{\theta \oplus \Lambda_T}(\cdot \mid x)$ is the memory-conditioned response distribution, and ℓ_{task} is a differentiable prediction loss that evaluates this prediction against the target y. Gradients propagate through the frozen backbone and decoder to update ψ, while θ, ξ, ϕ, and β remain fixed. The frozen compiler also allows the q_t representations to be precomputed during training. At deployment (Figure 1(c)), all modules are frozen. The first session initializes memory, and subsequent sessions update it through encoding and recurrent consolidation. Queries access the retained memory through the current backbone's decoder. Backbone replacement preserves the accumulated memory, session encoder, and learned gate, switching only to the new backbone and its adapted decoder. Subsequent sessions use the same memory space and consolidation policy.

3 Experiments

We evaluate three aspects of RPMem: cross-session memory performance and generalization, the contributions of its two training stages, and the lifecycle properties required for long-term deployment.

3.1 Experimental Setup

Benchmarks and metrics. PERMA (Liu et al., 2026b) evaluates memory formation, revision, and intervention across ordered sessions; we report its four clean and noisy single-domain (SD) and multi-domain (MD) settings and their macro-average. PersonaMem-v2 (Jiang et al., 2025) derives independent queries from long user histories; we report Overall, Self, and Current accuracy. PrefEval (Zhao et al., 2025) measures memory after 10, 70, and 300 intervening turns, averaged across its three information forms. Detailed task construction and metric definitions appear in Appendix E.1.

Baselines. Textual-memory methods include Full Context, RAG (Lewis et al., 2020) with M3-Embedding (Chen et al., 2024), Rolling Summary, Mem0 (Chhikara et al., 2025), and LightMem (Fang et al., 2026). Parametric baselines include SFT, trained on the same downstream split with the visible history, and the released Metis-9B recurrent-memory model (Zhang et al., 2026). No Context and Gold State provide query-only and annotated-memory references. Implementations and method-specific hyperparameters are documented in Appendix E.2.

Protocol. In the main comparison, RPMem and our baseline implementations use Qwen3-8B (Yang et al., 2025) with thinking disabled. We directly evaluate the released Metis-9B model. Generalization and transfer span five backbones across model families, scales, and architectures. PERMA uses ten-fold leave-one-user-out evaluation, while PersonaMem-v2 and PrefEval follow their official splits. For these benchmarks, we train the consolidation gate on each training split with the compiler from subsection 2.2 and answer-label cross-entropy. Full training and inference protocols appear in Appendix E.3.

3.2 Main Results

Table 1 compares all methods on cross-session memory benchmarks. Bold and underlined values mark the best and second-best non-oracle results, excluding Gold State.

Table 1: Main comparison on cross-session memory benchmarks (accuracy, %).

Type	Method	PERMA					PersonaMem-v2			PrefEval		
		SD-C	SD-N	MD-C	MD-N	Core	Overall	Self	Current	10	70	300
Reference	No Context	66.73	66.73	54.60	54.60	60.66	25.74	25.28	25.42	36.67	39.82	39.07
	Gold State	72.70	72.23	66.06	65.47	69.12	57.90	64.05	69.16	80.18	80.74	80.00
Textual	Rolling Summary	39.57	40.27	23.34	27.41	32.65	26.58	25.46	25.32	44.26	45.19	45.19
	LightMem	56.63	56.20	40.72	40.20	48.44	30.46	31.42	34.51	**81.30**	**79.07**	73.52
	Mem0	57.57	58.57	41.03	42.10	49.82	30.02	30.26	31.90	73.15	65.19	61.67
	RAG (BGE-M3)	63.00	62.53	49.16	47.46	55.54	30.32	31.13	31.70	59.26	54.63	50.74
	Full Context	76.47	73.03	70.72	69.95	72.54	31.10	32.72	33.97	60.00	45.00	40.56
Parametric	SFT	78.00	71.47	49.30	42.61	60.35	31.64	33.65	35.47	57.22	40.00	38.15
	Metis-9B	80.87	80.47	**80.17**	79.29	80.20	29.64	30.24	31.32	43.89	39.07	39.82
	RPMem (ours)	**90.57**	**91.20**	79.67	**80.62**	**85.52**	**40.22**	**43.61**	**47.28**	69.81	72.41	**74.81**

Across PERMA and PersonaMem-v2, RPMem obtains the strongest non-oracle aggregate performance under two independently constructed cross-session evaluations. On PERMA, it achieves 85.52%, outperforming Metis-9B and Full Context by 5.32 and 12.98 percentage points (pp), respectively, and ranks first in three of the four core settings. The advantage over Metis also holds with Qwen3.5-9B for both methods (Appendix H.1). RPMem also surpasses SFT by 25.17 pp under the same downstream split and visible-history boundary, extending its advantage to a parametric baseline trained on in-domain histories. RPMem also achieves the highest average across all seven historical-context variants at 86.53% (Appendix E.6). On PersonaMem-v2, it achieves 40.22% Overall accuracy, outperforming SFT by 8.58 pp and Full Context by 10.89 and 13.31 pp on Self and Current. The consistent advantage across benchmark constructions shows that the learned memory remains effective while tracking currently valid user information over long histories.

PrefEval further reveals how the methods respond as the intervening history grows. RPMem ranks third at 10 turns and second at 70 turns, then achieves the best 300-turn result at 74.81%, surpassing LightMem by 1.29 pp. Over the same range, LightMem declines from 81.30% to 73.52%, whereas RPMem rises from 69.81% to 74.81%. Its strongest relative performance at the longest interval shows that consolidated memory remains effective when the preference-bearing interaction is separated from the query by a long history.

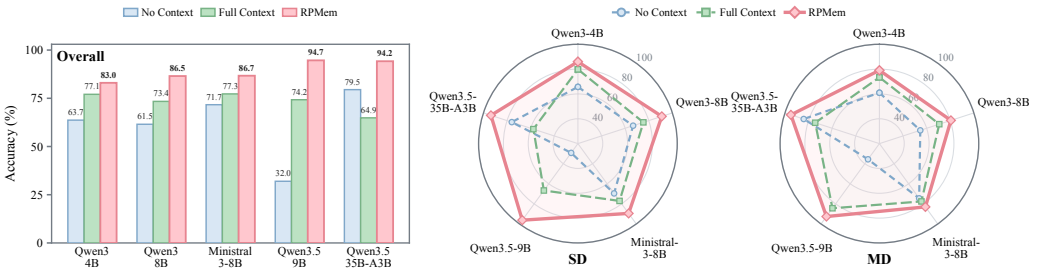


Figure 2: PERMA generalization across five backbones. Left: seven-variant average; right: SD and MD averages (accuracy, %).

Performance across language models. We evaluate RPMem on five backbones against matched No Context and Full Context references. Figure 2 shows consistent gains across model families, sizes, and dense and MoE architectures, with overall improvements of 5.95–20.41 pp over the stronger reference for each backbone. Full numerical results are reported in Appendix E.7. These findings support the generalizability of the memory mechanism across heterogeneous language-model backbones.

3.3 Ablation Studies

RPMem learns memory at two levels: the compiler encodes each session, and the recurrent consolidation module integrates the resulting memories over time. We isolate these components by varying the compiler objective under a fixed consolidation protocol and then varying the consolidation rule with the selected compiler. Table 2 reports both ablations with SD and MD macro-averages; complete definitions and setting-level results appear in Appendix D.

Compilation objective. Our compilation objective matches the context-conditioned reference distribution along fixed reference-response prefixes using forward KL (Fixed FKL). It reaches 85.52%, improving over selected-token cross-entropy (Top-K CE) by 7.86 pp and hard-label supervision (Compiler SFT) by 24.31 pp. The 16.45 pp gain from hard labels to selected-token probabilities shows that the relative likelihoods of plausible continuations carry information induced by the session. Retaining the probability mass outside the selected support contributes a further 7.86 pp, showing that faithful compilation also depends on distributional coverage. Sampled FKL remains within 1.77 pp of Fixed FKL, whereas reversing the KL direction reduces accuracy by more than 16 pp under sampled-response training. Together, these results favor matching the full context-conditioned distribution along a fixed, grounded response trajectory. Objective definitions and checkpoint-wise training dynamics are provided in Appendix D.1 and Appendix D.2.

Table 2: Ablations on PERMA (accuracy, %): **(a)** compilation objectives and **(b)** consolidation rules. Core SD and Core MD average the clean and noisy settings. FKL/RKL denote forward/reverse KL; bold and underline mark the best and second-best results in each panel.

Variant	Core SD	Core MD	Core Avg.
(a) Memory compilation objective			
Compiler SFT	72.20	50.23	61.21
Sampled RKL (top-K)	74.08	59.73	66.91
Sampled RKL (K3)	70.82	63.22	67.02
Top-K CE	82.99	72.34	77.66
Sampled FKL	<u>88.37</u>	<u>79.14</u>	<u>83.75</u>
Fixed FKL	**90.89**	**80.15**	**85.52**
(b) Cross-session consolidation			
Rank concatenation	14.97	12.71	13.84
Latent averaging	62.40	28.64	45.52
Latest Session	67.72	<u>39.40</u>	53.56
Factor averaging	<u>69.44</u>	38.99	<u>54.21</u>
Learned gate (ours)	**90.89**	**80.15**	**85.52**

Cross-session consolidation. The learned gate reaches 85.52%, exceeding the strongest evaluated fixed rule, LoRA factor averaging, by 31.31 pp. Factor averaging improves over retaining only the latest session by 0.65 pp, latent averaging performs 8.04 pp worse, and rank concatenation falls to 13.84% despite retaining every decoded session adapter. The learned gate gains 21.45 pp over factor averaging in SD histories and 41.16 pp in MD histories, localizing its largest benefit to sequences containing heterogeneous memory. These comparisons show the benefit of task-supervised, coordinate-wise integration over the evaluated fixed rules for combining accumulated and incoming information. Further definitions and data-efficiency results appear in Appendix D.3 and Appendix D.4. Additional experiments on Clean SD and Clean MD show consistent gains over update averaging, tuned EMA, and a task-trained scalar gate (Appendix D.3).

3.4 Lifecycle Properties

We first test the defining lifecycle claim that accumulated memory remains reusable after backbone replacement. We then evaluate its deployment viability through resource scaling and general-capability retention.

Cross-backbone reuse. We reuse the frozen Qwen3-8B session encoder with Qwen3-4B, Ministral-3-8B (Liu et al., 2026a), Qwen3.5-9B, and Qwen3.5-35B-A3B (Qwen Team, 2026), adapting only the model-specific decoder during compilation for each target. Under the same target-side training budget, transfer improves all 28 backbone–setting combinations over compilers trained from scratch and raises the cross-model average from 75.97% to 89.64%, showing that one parametric memory representation can serve different backbones through target-specific decoding. Complete results and adaptation costs appear in Appendix G.

Streaming deployment. We measure update latency, persistent-memory size, and historical query tokens as the session count increases from 1 to 64, comparing RPMem with Full Context, Mem0, and uncompressed adapter rank concatenation. Table 3 reports values at 64 sessions and growth over this range. Metric definitions and the full protocol are provided in Appendix E.5.

Table 3: Online deployment efficiency at 64 accumulated sessions.

Type	Method	Update		Memory		Hist. tokens ↓
		Time (s) ↓	Growth ↓	Size (MiB) ↓	Growth ↓	
Textual	Full Context	–	–	**0.138**	78.09×	28,498.4
	Mem0	12.554	1.10×	20.041	62.11×	441.7
Parametric	Adapter rank concatenation	0.948	29.46×	183.738	5.10×	**0**
	RPMem (ours)	**0.043**	1.03×	36.562	1.00×	**0**

At 64 sessions, RPMem writes a session in 0.043 seconds, approximately 22 times faster than rank concatenation and 292 times faster than Mem0. Across the full profile, update and memory growth remain $1.03\times$ and $1.00\times$, respectively, with no historical query tokens. Avoiding history reprocessing also reduces query cost: over the 5,000-question PersonaMem-v2 test set with approximately 32K-token histories, mean language-model forward time is 0.055 seconds, compared with 4.287 seconds for Full Context. Training the shared compiler requires an estimated 338 GPU-hours, whereas consolidation for a representative PERMA fold takes 13 minutes on one A800-80GB GPU; complete costs appear in Appendix E.5.

General-capability retention. Finally, we evaluate the unmodified backbone (Base LLM), Latest Session, and RPMem on MMLU (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), and IFEval (Zhou et al., 2023), averaging the two memory methods over ten user-specific adapters. Complete protocols and user-level results appear in Appendix E.5.

RPMem remains within 1.83 pp of Base LLM on MMLU, 1.43 pp on GSM8K, and 3.81 pp on IFEval, while improving over Latest Session on all three benchmarks. Cross-session memory therefore largely preserves the backbone's general capabilities.

Table 4: General-capability retention (accuracy, %).

Method	MMLU	GSM8K	IFEval
Base LLM	71.96	75.21	81.70
Latest Session	69.85	73.15	77.38
RPMem (ours)	70.13	73.78	77.89

4 Analysis of Learned Consolidation Dynamics

The experiments demonstrate the effectiveness of RPMem's cross-session memory. We then examine how the learned consolidation mechanism organizes memory as sessions accumulate. We study the Qwen3-8B configuration of RPMem under the PERMA ten-fold protocol. For every held-out user in Clean SD and Noisy SD, we replay the complete interaction history through the recurrent gate and trace how each session changes the memory, interacts with previously retained information, and survives subsequent updates.

4.1 Semantically Structured Memory Updates

At each session, the recurrent gate assigns a coordinate-wise retention coefficient to the accumulated memory, while the remaining weight determines how much of the incoming session memory is written. We summarize the complete update from one semantic event by its write magnitude, defined as the mean incoming-memory weight after composing all compiler segments in that event. We analyze fusion steps following direct first session initialization and compare write magnitudes between domain-emergence events, which establish a memory domain, and supplements, which add evidence to an existing domain. Emergence produces larger writes for every held-out user, and the emergence–supplement write difference is approximately five times larger than the difference between matched Clean and Noisy inputs (Figure 3(a–b)). Write magnitude therefore follows an event's contribution to the memory timeline while remaining stable under surface perturbations.

To examine domain structure, we measure the cosine similarity of centered write-weight vectors and the fraction of a historical source's contribution retained after one update. Same-domain events have more similar write patterns (Figure 3(c)), while a new event attenuates same-domain historical contributions more strongly than cross-domain contributions (Figure 3(d)). Both paired differences hold for every held-out user (Figure 3(e)), indicating that learned consolidation combines domain-specific revision with cross-domain preservation.

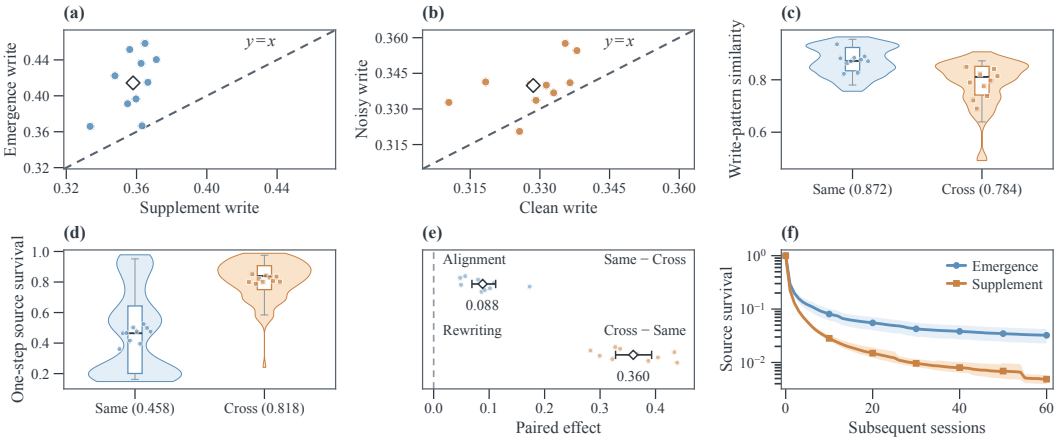


Figure 3: Learned consolidation dynamics on PERMA. (a–b) User-mean write magnitudes by event role and noise condition; diamonds mark across-user means, and dashed diagonals indicate equality. (c–d) Same- versus cross-domain write-pattern similarity and one-step source survival; violins show matched-pair distributions and points show user means. (e) Paired differences: Same minus Cross for similarity, Cross minus Same for survival. (f) Source survival over subsequent updates on a logarithmic scale. Statistical details, including interval definitions, appear in Appendix F.

4.2 Long-Term Survival of Memory Sources

We next examine whether these differentiated updates preserve memory over longer horizons. Unfolding the recurrence defines source survival as the fraction of a session's original contribution remaining after subsequent updates. The survival advantage of domain-emergence events grows from $1.31\times$ after one update to $6.64\times$ after 60 (Figure 3(f)). Functional retention is evaluated separately: across seven PERMA variants, accuracy is 86.7% immediately after the target memory event and 86.5% after subsequent intervening sessions (Appendix F). Across the observed histories, fixed-size memory exhibits differentiated writing and retention, with domain-establishing events receiving stronger writes and persisting longer than supplements. A representative held-out trajectory in Appendix J traces this organization through a complete interaction history.

5 Conclusion

We introduced RPMem, a recurrent parametric memory architecture that decouples accumulated experience from the serving backbone. By combining session compilation with learned consolidation, it maintains fixed-size cross-session memory without adding historical text to the query context. Experiments on three memory benchmarks demonstrate strong performance, while ablations and memory-dynamics analyses support the role of selective integration within a fixed capacity. Evaluations across five backbones demonstrate broad applicability, and backbone-replacement experiments show that accumulated memory can be reused through model-specific decoding. Deployment measurements further indicate approximately constant update cost and per-user memory size as histories grow, with limited degradation in general capabilities. Together, these results support lifecycle-independent parametric memory that accumulates experience across sessions and remains usable as serving backbones evolve.

6 Limitations

Application-specific consolidation aligns memory retention and updating with the target scenario's requirements, but the learned policy's applicability to scenarios with substantially different memory needs remains to be studied. Future work could explore cross-scenario learning that balances shared consolidation capabilities with application-specific adaptation.

AI USE STATEMENT

We used generative AI tools for manuscript polishing, LaTeX typesetting, and selected code implementation. The manuscript was collaboratively written and reviewed by multiple authors. All AI-assisted code underwent review by at least two authors. We take responsibility for the final content of this work, including all AI-assisted text and code.

ETHICS STATEMENT

This work studies long-term agent memory using existing datasets and derived training examples, with data construction documented in Appendix C. Persistent memory can retain sensitive user information and carry biases from historical interactions into future responses. Deployment with personal data should therefore incorporate informed user consent, access controls, and mechanisms for memory correction and deletion. Any redistribution of source or derived data should respect the applicable licenses and privacy requirements.

REPRODUCIBILITY STATEMENT

The architecture and learning objectives are specified in section 2, with pseudocode for both training stages and deployment in Appendix B. Appendix C documents the data sources, preprocessing, query–response construction, and downstream splits. Appendix E provides benchmark protocols, baseline implementations, model configurations, and training hyperparameters. Additional procedures for ablations, memory-dynamics analysis, and backbone transfer are reported in Appendix D, Appendix F, and Appendix G, respectively. Our implementation is available in an anonymous repository at `https://anonymous.4open.science/r/rpmem-submission-B176`.

REFERENCES

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *Advances in Neural Information Processing Systems*, volume 38, pp. 113506–113543. Curran Associates, Inc., 2025. doi: 10.52202/085713-3786.

Jiaqi Cao, Jiarui Wang, Rubin Wei, Qipeng Guo, Kai Chen, Bowen Zhou, and Zhouhan Lin. Memory Decoder: A pretrained, plug-and-play memory for large language models. In *Advances in Neural Information Processing Systems*, volume 38, pp. 115487–115510. Curran Associates, Inc., 2025. doi: 10.52202/085713-3851.

Ruike Cao, Shaojie Bai, Fugen Yao, Liang Dong, Jian Xu, and Li Xiao. ATPO: Adaptive tree policy optimization for multi-turn medical dialogue. In *International Conference on Learning Representations*, volume 2026, pp. 31272–31292. OpenReview.net, 2026. URL `https://openreview.net/forum?id=2bv3B8B9bl`.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-LoRA: Learning to instantly internalize contexts. In *Forty-third International Conference on Machine Learning*, 2026. URL `https://openreview.net/forum?id=iW1oBBO72S`.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 2318–2335. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.137.

Tong Chen, Hao Fang, Patrick Xia, Xiaodong Liu, Benjamin Van Durme, Luke Zettlemoyer, Jianfeng Gao, and Hao Cheng. Generative Adapter: Contextualizing language models in parameters with A single forward pass. In *International Conference on Learning Representations*, volume 2025, pp. 27450–27470. OpenReview.net, 2025. URL `https://openreview.net/forum?id=bc3sUsS6ck`.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. In *ECAI 2025—28th European Conference on Artificial Intelligence*, volume 413 of *Frontiers in Artificial Intelligence and Applications*, pp. 2993–3000. IOS Press, 2025. doi: 10.3233/FAIA251160.

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

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. LightMem: Lightweight and efficient memory-augmented generation. In *International Conference on Learning Representations*, volume 2026, pp. 98706–98729. OpenReview.net, 2026. URL https://openreview.net/forum?id=dyJ0GWpjJB.

Naibin Gu, Peng Fu, Xiyu Liu, Ke Ma, Zheng Lin, and Weiping Wang. Adapt once, thrive with updates: Transferable parameter-efficient fine-tuning on evolving base models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14765–14783, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.719.

Haoliang Han. Memory depth, not memory access: Selective parametric consolidation for long-running language agents. *CoRR*, abs/2606.26806, 2026. URL https://arxiv.org/abs/2606.26806.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*. OpenReview.net, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of AI agents. *CoRR*, abs/2512.13564, 2025. URL https://arxiv.org/abs/2512.13564.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and João Carreira. Perceiver: General perception with iterative attention. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4651–4664. PMLR, 2021. URL https://proceedings.mlr.press/v139/jaegle21a.html.

Suozhao Ji, Baodong Wu, Zehao Wang, Lei Xia, Qingping Li, Ruisong Wang, Wenbo Ding, Zhenhua Zhu, Boxun Li, Guohao Dai, and Yu Wang. Infini Memory: Maintainable topic documents for long-term LLM agent memory. *CoRR*, abs/2606.10677, 2026. URL https://arxiv.org/abs/2606.10677.

Bowen Jiang, Yuan Yuan, Maohao Shen, Zhuoqun Hao, Zhangchen Xu, Zichen Chen, Ziyi Liu, Anvesh Rao Vijjini, Jiashu He, Hanchao Yu, Radha Poovendran, Gregory Wornell, Lyle Ungar, Dan Roth, Sihao Chen, and Camillo Jose Taylor. PersonaMem-v2: Towards personalized intelligence via learning implicit user personas and agentic memory. *CoRR*, abs/2512.06688, 2025. URL https://arxiv.org/abs/2512.06688.

ChanJoo Jung and Jaehyung Kim. TiTok: Transfer token-level knowledge via contrastive excess to transplant LoRA. In *International Conference on Learning Representations*, volume 2026, pp. 123084–123113. OpenReview.net, 2026. URL https://openreview.net/forum?id=0B5K9pIdSK.

Yuri Kuratov, Matvey Kairov, Aydar Bulatov, Ivan Rodkin, and Mikhail Burtsev. GradMem: Learning to write context into memory with test-time gradient descent. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=hoLdhnkP0P.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Jiaang Li, Quan Wang, Zhongnan Wang, Yongdong Zhang, and Zhendong Mao. ELDER: Enhancing lifelong model editing with Mixture-of-LoRA. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24440–24448. AAAI Press, 2025a. doi: 10.1609/aaai.v39i23.34622.

Zhiyu Li, Shichao Song, Hanyu Wang, Simin Niu, Ding Chen, Jiawei Yang, Chenyang Xi, Huayi Lai, Jihao Zhao, Yezhaohui Wang, Junpeng Ren, Zehao Lin, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhiqiang Yin, Qingchen Yu, Bo Tang, Hongkang Yang, Zhi-Qin John Xu, and Feiyu Xiong. MemOS: An operating system for memory-augmented generation (MAG) in large language models. *CoRR*, abs/2505.22101, 2025b. URL https://arxiv.org/abs/2505.22101.

Alexander H. Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, Alexandre Sablayrolles, Amélie Héliou, Amos You, Andy Ehrenberg, Andy Lo, Anton Eliseev, Antonia Calvi, Avinash Sooriyarachchi, Baptiste Bout, Baptiste Rozière, Baudouin De Monicault, Clémence Lanfranchi, Corentin Barreau, Cyprien Courtot, Daniele Grattarola, Darius Dabert, Diego de las Casas, Elliot Chane-Sane, Faruk Ahmed, Gabrielle Berrada, Gaëtan Ecrepont, Gauthier Guinet, Georgii Novikov, Guillaume Kunsch, Guillaume Lample, Guillaume Martin, Gunshi Gupta, Jan Ludziejewski, Jason Rute, Joachim Studnia, Jonas Amar, Joséphine Delas, Josselin Somerville Roberts, Karmesh Yadav, Khyathi Chandu, Kush Jain, Laurence Aitchison, Laurent Fainsin, Léonard Blier, Lingxiao Zhao, Louis Martin, Lucile Saulnier, Luyu Gao, Maarten Buyl, Margaret Jennings, Marie Pellat, Mark Prins, Mathieu Poirée, Mathilde Guillaumin, Matthieu Dinot, Matthieu Futeral, Maxime Darrin, Maximilian Augustin, Mia Chiquier, Michel Schimpf, Nathan Grinsztajn, Neha Gupta, Nikhil Raghuraman, Olivier Bousquet, Olivier Duchenne, Patricia Wang, Patrick von Platen, Paul Jacob, Paul Wambergue, Paula Kurylowicz, Pavankumar Reddy Muddireddy, Philomène Chagniot, Pierre Stock, Pravesh Agrawal, Quentin Torroba, Romain Sauvestre, Roman Soletskyi, Rupert Menneer, Sagar Vaze, Samuel Barry, Sanchit Gandhi, Siddhant Waghjale, Siddharth Gandhi, Soham Ghosh, Srijan Mishra, Sumukh Aithal, Szymon Antoniak, Teven Le Scao, Théo Cachet, Theo Simon Sorg, Thibaut Lavril, Thiziri Nait Saada, Thomas Chabal, Thomas Foubert, Thomas Robert, Thomas Wang, Tim Lawson, Tom Bewley, Tom Edwards, Umar Jamil, Umberto Tomasini, Valeriia Nemychnikova, Van Phung, Vincent Maladière, Virgile Richard, Wassim Bouaziz, Wen-Ding Li, William Marshall, Xinghui Li, Xinyu Yang, Yassine El Ouahidi, Yihan Wang, Yunhao Tang, and Zaccharie Ramzi. Ministral 3. *CoRR*, abs/2601.08584, 2026a. URL https://arxiv.org/abs/2601.08584.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638.

Shuochen Liu, Junyi Zhu, Long Shu, Junda Lin, Yuhao Chen, Haotian Zhang, Chao Zhang, Derong Xu, Jia Li, Bo Tang, Zhiyu Li, Feiyu Xiong, Enhong Chen, and Tong Xu. PERMA: Benchmarking personalized memory agents via event-driven preference and realistic task environments. *CoRR*, abs/2603.23231, 2026b. URL https://arxiv.org/abs/2603.23231.

Yewei Liu, Xiyuan Wang, Yansheng Mao, Yoav Gelberg, Haggai Maron, and Muhan Zhang. SHINE: A scalable in-context hypernetwork for mapping context to LoRA in a single pass. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026c. URL https://openreview.net/forum?id=ZMexYcAibv.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *CoRR*, abs/2310.08560, 2023. URL https://arxiv.org/abs/2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 2:1–2:22. ACM, 2023. doi: 10.1145/3586183.3606763.

Qwen Team. Qwen3.5: Towards native multimodal agents, 2026. URL https://qwen.ai/blog?id=qwen3.5.

Tao Ren, Weiyao Luo, Hui Yang, Rongzhi Zhu, Xiang Huang, Yuchuan Wu, Bingxue Chou, Jieping Ye, Jiafeng Liang, Yongbin Li, and Yijie Peng. Scaling self-evolving agents via parametric memory. *CoRR*, abs/2606.04536, 2026. URL https://arxiv.org/abs/2606.04536.

Leheng Sheng, Yongtao Zhang, Wenchang Ma, Yaorui Shi, Ting Huang, Xiang Wang, An Zhang, Ke Shen, and Tat-Seng Chua. When to memorize and when to stop: Gated recurrent memory for long-context reasoning. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=mOPwfQSfJq.

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

Jihoon Tack, Jaehyung Kim, Eric Mitchell, Jinwoo Shin, Yee Whye Teh, and Jonathan Richard Schwarz. Online adaptation of language models with a memory of amortized contexts. In *Advances in Neural Information Processing Systems*, volume 37, pp. 130109–130135. Curran Associates, Inc., 2024. doi: 10.52202/079017-4134.

Zhaoxuan Tan, Zixuan Zhang, Haoyang Wen, Zheng Li, Rongzhi Zhang, Pei Chen, Fengran Mo, Zheyuan Liu, Qingkai Zeng, Qingyu Yin, and Meng Jiang. Instant personalized large language model adaptation via hypernetwork. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23557–23580, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.1081.

Zhenheng Tang, Fanjunduo Wei, Zichen Tang, Peijie Dong, Xiang Liu, Qian Wang, Xiaowen Chu, and Bo Li. Parameters as agentic memory: Internalizing long-horizon memories for efficient LLM agents. In *Second Workshop on Agents in the Wild: Safety, Security, and Beyond at ICML*, 2026. URL https://openreview.net/forum?id=ptIjkWmtl9.

Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. WISE: Rethinking the knowledge memory for lifelong model editing of large language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 53764–53797. Curran Associates, Inc., 2024a. doi: 10.52202/079017-1703.

Runqian Wang, Soumya Ghosh, David Cox, Diego Antognini, Aude Oliva, Rogério Feris, and Leonid Karlinsky. Trans-LoRA: towards data-free transferable parameter efficient finetuning. In *Advances in Neural Information Processing Systems*, volume 37, pp. 61217–61237. Curran Associates, Inc., 2024b. doi: 10.52202/079017-1957.

Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. MEMORYLLM: Towards self-updatable large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50453–50466. PMLR, 2024c. URL https://proceedings.mlr.press/v235/wang24s.html.

Yu Wang, Dmitry Krotov, Yuanzhe Hu, Yifan Gao, Wangchunshu Zhou, Julian McAuley, Dan Gutfreund, Rogério Feris, and Zexue He. M+: Extending MemoryLLM with scalable long-term memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63308–63323. PMLR, 2025a. URL https://proceedings.mlr.press/v267/wang25au.html.

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. In *International Conference on Learning Representations*, volume 2025, pp. 16961–16979. OpenReview.net, 2025b. URL https://openreview.net/forum?id=aCPFCDL9QY.

Rubin Wei, Jiaqi Cao, Jiarui Wang, Jushi Kai, Qipeng Guo, Bowen Zhou, and Zhouhan Lin. MLP Memory: A retriever-pretrained memory for large language models. In *International Conference on Learning Representations*, volume 2026, pp. 132772–132795. OpenReview.net, 2026. URL https://openreview.net/forum?id=1SMdxRtLBp.

Yicheng Xiao, Lin Song, Rui Yang, Cheng Cheng, Yixiao Ge, Xiu Li, and Ying Shan. LoRA-Gen: Specializing large language model via online LoRA generation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 68459–68471. PMLR, 2025. URL https://proceedings.mlr.press/v267/xiao25e.html.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, pp. 17577–17604. Curran Associates, Inc., 2025. doi: 10.52202/085713-0593.

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. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.583.

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

Tianjun Yao, Yongqiang Chen, Yujia Zheng, Pan Li, Zhiqiang Shen, and Kun Zhang. ParamMem: Augmenting language agents with parametric reflective memory. *CoRR*, abs/2602.23320, 2026. URL https://arxiv.org/abs/2602.23320.

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. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21457–21483. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.981.

Zeyu Zhang, Ziliang Guo, Yihang Sun, Xichong Zhang, Xixuan Hao, Zehao Lin, Yang Zhang, Xiaoyan Zhao, Tong Shen, Bo Tang, Zhi-Qin John Xu, Junchi Yan, Haofen Wang, Xu Chen, Feiyu Xiong, Zhiyu Li, and Tat-Seng Chua. Metis: Memory foundation model. *CoRR*, abs/2607.26760, 2026. URL https://arxiv.org/abs/2607.26760.

Siyan Zhao, Mingyi Hong, Yang Liu, Devamanyu Hazarika, and Kaixiang Lin. Do LLMs recognize your preferences? evaluating personalized preference following in LLMs. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*, volume 2025, pp. 15888–15931. OpenReview.net, 2025. URL https://openreview.net/forum?id=QWunLKbBGF.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731. AAAI Press, 2024. doi: 10.1609/aaai.v38i17.29946.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *CoRR*, abs/2311.07911, 2023. URL https://arxiv.org/abs/2311.07911.

APPENDIX

A RELATED WORK

Text-Based Agent Memory. Most agent-memory systems maintain interaction history as external textual records and retrieve selected entries for each query. Generative Agents (Park et al., 2023) organizes experience as a memory stream, MemGPT (Packer et al., 2023) manages hierarchical storage, and Mem0 and LightMem (Chhikara et al., 2025; Fang et al., 2026) extract structured records from conversations. A-MEM (Xu et al., 2025) constructs linked notes, while Infini Memory (Ji et al., 2026) consolidates observations into topic documents. Memory-R1 and AgeMem (Yan et al., 2026; Yu et al., 2026) learn memory management through reinforcement learning, with AgeMem jointly controlling external memory and the active context. GRU-Mem (Sheng et al., 2026) learns when to update textual memory and stop reading further context. At the infrastructure level, MemOS (Li et al., 2025b) unifies the organization and lifecycle management of textual, activation, and parametric memories; RPMem addresses the learned representation and integration of cross-session parametric memory. These advances improve storage and retrieval, yet cross-session knowledge must still be selected and reconstructed within a finite context on every call, making long-horizon performance jointly dependent on retrieval quality, compression fidelity, and contextual reasoning.

Parametric Agent Memory. Parametric memory makes prior information available through model computation rather than context injection. PAM (Tang et al., 2026) internalizes long-horizon agent experience into LoRA adapters through fine-tuning; TMEM (Ren et al., 2026) accumulates online LoRA gradient updates from extracted experience. SELF-PARAM (Wang et al., 2025b) integrates successive contexts into model weights by matching context-conditioned response distributions. WISE and ELDER (Wang et al., 2024a; Li et al., 2025a) manage sequential knowledge edits through routed side memory and mixtures of LoRA adapters, respectively. EVAF (Han, 2026) selects events using surprise and valence signals, then updates a LoRA adapter with buffered replay. These approaches use gradient-based writes; RPMem learns compilation and consolidation offline and performs subsequent memory updates through forward computation. ParamMem (Yao et al., 2026) encodes cross-sample reflection patterns into model parameters. MLP Memory (Wei et al., 2026) distills a kNN retriever's output distribution and combines its predictions with the language model's. Titans (Behrouz et al., 2025) updates a neural memory module at test time, while GradMem (Kuratov et al., 2026) optimizes prefix memory tokens using a context-reconstruction objective with the language model frozen. Architecture-native methods maintain mutable memory inside the language model: MEMORYLLM (Wang et al., 2024c) introduces a self-updatable latent memory pool, M+ (Wang et al., 2025a) extends it with scalable long-term retrieval, and Metis (Zhang et al., 2026) learns forward-updated memory matrices through memory-specific mid-training. Metis represents the closest concurrent work: it satisfies both forward-only writing and cross-session evolution through gated memory dynamics, yet its memory dimensions, update projections, and read-write mechanisms are defined by the specific backbone architecture, preventing memory reuse after model replacement. These architecture-native approaches couple memory to the backbone that produced it.

Context-to-Adapter Generation and Transfer. Context distillation optimizes parameters to reproduce behavior induced by a given context (Snell et al., 2022). MAC (Tack et al., 2024) amortizes documents into a memory bank and aggregates their modulations for each query. RPMem consolidates incoming sessions into persistent, fixed-size memory before a query arrives. Doc-to-LoRA (Charakorn et al., 2026) amortizes document-specific distillation into a shared Perceiver hypernetwork that generates LoRA parameters in one forward pass; SHINE (Liu et al., 2026c) scales this mapping to multi-layer adapters, and Profile-to-PEFT (Tan et al., 2026) generates personalized adapters from textual user profiles. These methods establish forward mappings from bounded text to model parameters, but each updated source produces a new static adapter without a mechanism for cross-session memory evolution. Generative Adapter (Chen et al., 2025) supports forward-only adaptation to streaming context through backbone-specific weight updates, and LoRA-Gen (Xiao et al., 2025) generates adapters for a target model from task descriptions. Trans-LoRA, Trans-PEFT, and TiTok (Wang et al., 2024b; Gu et al., 2025; Jung & Kim, 2026) transfer adapters learned from fixed task data across backbones, addressing model portability for static adapters rather than evolving memory. Memory Decoder (Cao et al., 2025) learns a retriever-derived prediction distribution that can be combined with backbones sharing a tokenizer, enabling cross-model reuse of domain

memory. RPMem complements this portability with recurrent cross-session integration and model-specific decoding of accumulated memory into LoRA parameters.

B ALGORITHMS

The following algorithms specify the two training stages and the online memory interface using the notation of section 2. The context encoder E_ξ and answering backbone f_θ are frozen throughout. Compilation optimizes the Perceiver-based resampler parameters ϕ and decoder parameters β; consolidation subsequently optimizes the gate parameters $\psi = \{W_g, b_g\}$. The training data are constructed in Appendix C, and numerical configurations are given in Appendix E.4. We write $\mathrm{AdamW}(w, g; s)$ for an optimizer step with parameters w, gradient g, and optimizer state s, returning the updated parameters and state. The state contains the moment estimates and step count and is initialized to zero. All optimizer hyperparameters follow the configuration appendix.

Session compilation. Let $\mathcal{D}_{\mathrm{comp}}$ denote the collection of training sessions and their fixed query–response pairs. For each pair, the reference model receives the session as text and is evaluated on the fixed response prefix. In Algorithm B.1, $\mathrm{TopK}_v(p, K)$ returns the indices of the K largest probabilities in distribution p. The projection $\Pi_S(p) = ((p(v))_{v\in S},\ 1-\sum_{v\in S} p(v))$ collects the probabilities on support S and their complementary tail mass. The cache $\mathcal{T}(c, i, j)$ associates session c, pair i, and response position j with this support and projected reference distribution; its probabilities are recovered from the stored reference log-probabilities. We denote the cached projected distribution by $\bar{p}^{\mathrm{ctx}}_{i,j}$. The symbols $\mathcal{B}$, $\mathcal{L}_c$, and $\widehat{\mathcal{L}}_{\mathrm{comp}}$ denote a mini-batch, its individual session objectives, and their mean, respectively. The number of compilation epochs is denoted by E_{comp}; the reference cache is prepared once and reused across these epochs. Factor regularization $\mathcal{R}_1$ follows Equation E.1.

Algorithm B.1 Single-session memory compilation

Require: $\mathcal{D}_{\mathrm{comp}}, E_\xi, f_\theta, G_\phi, D_\beta, K, \lambda, E_{\mathrm{comp}}$
Ensure: ϕ, β

$s_{\mathrm{comp}} \leftarrow 0$
for $(c, \{(x_i, y_i)\}_{i=1}^{Q}) \in \mathcal{D}_{\mathrm{comp}}$ **do**
  for $i = 1, \dots, Q;\ j = 1, \dots, N_i$ **do**
    $p^{\mathrm{ctx}}_{i,j} \leftarrow p_\theta(\cdot \mid c, x_i, y_{i,<j})$
    $S_{i,j} \leftarrow \mathrm{TopK}_v(p^{\mathrm{ctx}}_{i,j}, K)$
    $\mathcal{T}(c, i, j) \leftarrow (S_{i,j}, \Pi_{S_{i,j}}(p^{\mathrm{ctx}}_{i,j}))$
  end for
end for
for $e = 1, \dots, E_{\mathrm{comp}}$ **do**
  for each mini-batch $\mathcal{B} \subset \mathcal{D}_{\mathrm{comp}}$ **do**
    for $(c, \{(x_i, y_i)\}_{i=1}^{Q}) \in \mathcal{B}$ **do**
      $q \leftarrow G_\phi(E_\xi(c))$
      $\Lambda(c) \leftarrow D_\beta(q)$
      for $i = 1, \dots, Q;\ j = 1, \dots, N_i$ **do**
        $(S_{i,j}, \bar{p}^{\mathrm{ctx}}_{i,j}) \leftarrow \mathcal{T}(c, i, j)$
        $p^{\mathrm{mem}}_{i,j} \leftarrow p_{\theta\oplus\Lambda(c)}(\cdot \mid x_i, y_{i,<j})$
        $\ell_{i,j} \leftarrow D_{\mathrm{KL}}(\bar{p}^{\mathrm{ctx}}_{i,j} \,\|\, \Pi_{S_{i,j}}(p^{\mathrm{mem}}_{i,j}))$
      end for
      $\mathcal{L}_c \leftarrow Q^{-1} \sum_{i=1}^{Q} N_i^{-1} \sum_{j=1}^{N_i} \ell_{i,j} + \lambda \mathcal{R}_1(c)$
    end for
    $\widehat{\mathcal{L}}_{\mathrm{comp}} \leftarrow |\mathcal{B}|^{-1} \sum_{c\in\mathcal{B}} \mathcal{L}_c$
    $((\phi, \beta), s_{\mathrm{comp}}) \leftarrow \mathrm{AdamW}((\phi, \beta), \nabla_{\phi,\beta} \widehat{\mathcal{L}}_{\mathrm{comp}}; s_{\mathrm{comp}})$
  end for
end for

Cross-session consolidation. Training examples are drawn from the target application's distribution $\mathcal{D}_{\mathrm{task}}$. For each example $(\mathcal{C}_{1:T}, x, y)$, the frozen session encoder produces ordered memories $q_1, \dots, q_T$. These tensors may be cached and shared by examples with the same history segments. We denote the ordered cache for a history by $\mathcal{Q}(\mathcal{C}_{1:T})$, the configured initial gate by ψ_0, and the number of training epochs by E_{con}. The operator $\mathrm{Shuffle}$ permutes training examples using the configured random seed. Algorithm B.2 resets the recurrent memory for each example, initializes it

from the first session, and differentiates the task loss through the subsequent recurrence. The symbol $\widehat{\ell}$ denotes the loss of the current example. Freezing decoder and backbone parameters preserves differentiation with respect to their memory-conditioned inputs.

Algorithm B.2 Application-specific cross-session memory consolidation

Require: $\mathcal{D}_{\text{task}}, E_\xi, G_\phi, D_\beta, f_\theta, \ell_{\text{task}}, \psi_0, E_{\text{con}}$
Ensure: $\psi = \{W_g, b_g\}$
$\psi \leftarrow \psi_0;\ s_{\text{con}} \leftarrow 0$
for $(\mathcal{C}_{1:T}, x, y) \in \mathcal{D}_{\text{task}}$ **do**
  $\mathcal{Q}(\mathcal{C}_{1:T}) \leftarrow (G_\phi(E_\xi(c_t)))_{t=1}^{T}$
end for
for $e = 1, \ldots, E_{\text{con}}$ **do**
  for $(\mathcal{C}_{1:T}, x, y) \in \text{Shuffle}(\mathcal{D}_{\text{task}})$ **do**
    $(q_1, \ldots, q_T) \leftarrow \mathcal{Q}(\mathcal{C}_{1:T})$
    $h_0 \leftarrow 0;\ h_1 \leftarrow q_1$
    for $t = 2, \ldots, T$ **do**
      $z_t \leftarrow \sigma([h_{t-1}; q_t]W_g + b_g)$
      $h_t \leftarrow z_t \odot h_{t-1} + (1 - z_t) \odot q_t$
    end for
    $\Lambda_T \leftarrow D_\beta(h_T)$
    $\widehat{\ell} \leftarrow \ell_{\text{task}}(p_{\theta \oplus \Lambda_T}(\cdot \mid x), y)$
    $(\psi, s_{\text{con}}) \leftarrow \text{AdamW}(\psi, \nabla_\psi \widehat{\ell}; s_{\text{con}})$
  end for
end for

Online memory and backbone replacement. Algorithm B.3 separates arrival of a session from answering a query. All module parameters are fixed during deployment. The retained memory h_T also serves as the input to an adapted decoder $D_{\beta'}$ for a target backbone $f_{\theta'}$. Decoder adaptation uses the compilation objective with target-backbone reference distributions, holding the memory encoder and target backbone fixed, as described in subsection 2.2. This adaptation precedes deployment of the replacement backbone; the accumulated memory, session encoder, and learned gate are retained across the switch. The target decoder maps the unchanged memory representation to the new backbone's layer and module dimensions. The stream $\mathcal{E}$ contains typed events $\text{Session}(c)$, $\text{Query}(x)$, and $\text{Replace}(D_{\beta'}, f_{\theta'})$. The operators Decode and Emit generate a response under the chosen decoding policy and return it to the caller, respectively. Queries in this algorithm follow the first memory write.

Algorithm B.3 Streaming memory updates and query-time decoding

Require: $E_\xi, G_\phi, D_\beta, f_\theta, \psi, \mathcal{E}$
$T \leftarrow 0;\ h_0 \leftarrow 0$
for $e \in \mathcal{E}$ **do**
  if $e = \text{Session}(c)$ **then**
    $q \leftarrow G_\phi(E_\xi(c))$
    if $T = 0$ **then**
      $h_{T+1} \leftarrow q$
    else
      $z \leftarrow \sigma([h_T; q]W_g + b_g)$
      $h_{T+1} \leftarrow z \odot h_T + (1 - z) \odot q$
    end if
    $T \leftarrow T + 1$
  else if $e = \text{Query}(x)$ **then**
    $\Lambda_T \leftarrow D_\beta(h_T)$
    $y \leftarrow \text{Decode}(p_{\theta \oplus \Lambda_T}(\cdot \mid x))$
    $\text{Emit}(y)$
  else if $e = \text{Replace}(D_{\beta'}, f_{\theta'})$ **then**
    $(D_\beta, f_\theta) \leftarrow (D_{\beta'}, f_{\theta'})$ ▷ h_T, ψ unchanged
  end if
end for

C TRAINING DATA CONSTRUCTION

Compilation trains the memory compiler to encode individual sessions in latent memory, and consolidation learns a memory-integration policy for a target application scenario. The two stages use separate data constructions: query–response pairs for individual sessions, followed by ordered histories paired with task queries and targets. Their training procedures are specified in Appendix B.

C.1 SINGLE-SESSION COMPILATION DATA

Sources and normalization. The source mixture covers conversational interactions, task-oriented and tool-use dialogues, coding-agent trajectories, and synthetic QA. Table C.1 lists the configured sources and domain-level sampling targets. These percentages are token-budget targets used during construction. The final corpus is filtered jointly across sources.

Table C.1: Compilation sources and target shares of context tokens.

Domain	Sources	Target share
Conversation	UltraChat-200k	35%
Task and tool use	Taskmaster-2; Schema-Guided Dialogue; Toucan SFT	25%
Coding	SWE-Zero OpenHands trajectories	20%
Synthetic QA	D2L synthetic QA, selected level-1 training shards	20%

Source adapters map records to an ordered event stream. Each event retains its identifier, speaker role, event type, and content; tool calls additionally retain available names, arguments, and results. This representation preserves the evidence needed to associate each pair with the corresponding interaction. For Toucan and SWE-Zero, normalization removes system messages, assistant reasoning fields, and designated reasoning-tool calls. The synthetic source is decoded with its source tokenizer before conversion to the same schema. Long records are segmented at event boundaries under the 4,096-token budgets of both the Qwen3-8B and ModernBERT tokenizers, with one event of overlap between adjacent segments.

Filtering. Duplicate detection applies Unicode normalization and case folding before hashing word-token sequences. Exact duplicates share a SHA-256 hash; near-duplicate detection uses 64-bit SimHash over three-token shingles, rejecting signatures within Hamming distance three for texts containing at least 20 normalized tokens. Filtering is applied across the accepted corpus before the training/validation assignment. The source configuration excludes designated downstream benchmark sources, and coding trajectories are filtered against an explicit evaluation-repository exclusion list. Source identifiers, revisions, and record provenance are retained in the data manifests.

Table C.2 lists the repositories and revision prefixes specified in the source configuration. The coding exclusion list contains the 100 repositories represented by the 500 tasks in the verified split of `SWE-bench-Live/SWE-bench-Live`, revision `a637bd46829f`. This repository-level filter is applied during source normalization. The synthetic subset selects level-1 training shards and decodes their token sequences with the Mistral-7B-Instruct-v0.2 tokenizer.

Table C.2: Configured compilation data sources. Revision identifiers are abbreviated to 12 characters.

Repository and subset	Revision
`HuggingFaceH4/ultrachat_200k`	`8049631c405a`
`DeepPavlov/TaskMaster2`	`333d425dd1f5`
`google-research-datasets/dstc8-schema-guided-dialogue`, training dialogues	`e852981ae349`
`Agent-Ark/Toucan-1.5M`, SFT subset	`0df3cf37f2ab`
`nvidia/SWE-Zero-openhands-trajectories`	`7b3cd106d00f`
`SakanaAI/self_gen_qa_d2l`, level-1 training shards	Selected shards

Query–response pairs. Every accepted session has exactly ten query–response pairs. A pair contains a query and a fixed reference response; answerable pairs also cite supporting session events.

Existing source-grounded pairs are preserved, and missing pairs are generated and validated against the same evidence-linking contract. For answerable questions, the generation prompt requests concise responses grounded in explicit event content, with evidence identifiers copied from the session. The preparation pipeline also supports source-derived response pairs and deterministic event-recall fallbacks to complete pair coverage. Validation checks the pair schema, nonempty reference responses, answerability flags, evidence identifiers, exact pair counts, and preservation of the underlying session content. These checks establish structural and evidence-link consistency; reference responses remain the fixed trajectories used in distribution matching.

Table C.3 specifies the documented reproduction configuration for query–response generation. The generator returns a compact JSON record containing the query, question type, answerability flag, reference response, and evidence-event identifiers. Answerable pairs must cite valid events; the protocol also permits unanswerable pairs with an empty evidence list. All retained pairs receive the same training treatment, without answerability-based filtering or weighting at training time. The parser removes duplicate query–response pairs and validates each accepted pair against its session. Generation continues for sessions with incomplete coverage, up to the configured round limit. Event-grounded fallbacks then complete the ten pairs, and a final per-source validation checks coverage and preservation of session content. Source-derived and fallback pairs follow the same schema as generated pairs. Prompt examples are provided in Appendix C.3.

Table C.3: Documented query–response generation configuration.

Setting	Value
Generator	Locally hosted Qwen3-8B
Generation policy	`grounded_compact_v3`
Pairs per session	10
Temperature	0
Maximum generated tokens per request	768
Generation batch size	64
Maximum generation rounds	3
Additional source-derived pairs per session	0 by default
Event-grounded fallback	Enabled

Corpus freeze and splits. The resulting frozen corpus contains 671,030 sessions, 6,710,300 pairs, and 525,898,190 backbone-tokenized context tokens. Its deterministic split assigns eight pairs from each of 664,128 sessions to training, reserves two disjoint pairs from those sessions for query-level validation, and holds out another 6,902 sessions for session-level validation. The session-level assignment uses a seeded hash of the session identifier, with a 1% holdout probability. Within training sessions, a second seeded hash orders pair identifiers to select the two query-held-out pairs. Both assignments use seed 42. Session-level validation evaluates unseen session content, whereas query-level validation evaluates disjoint pairs over training-session content.

Table C.4 gives the pair counts for each partition. Training and query-level validation share session content and use disjoint pair sets; session-level validation is disjoint in session content.

Table C.4: Compilation data partitions.

Partition	Sessions	Pairs per session	Total pairs
Training	664,128	8	5,313,024
Query-level validation	664,128	2	1,328,256
Session-level validation	6,902	10	69,020

Reference probability targets. For every fixed reference response, the frozen base language model is evaluated with the corresponding session present in its textual context. At each response position, the data pipeline stores the model's top-K token log-probabilities; the remaining mass is reconstructed as the tail category in Equation 9. The compiler is trained from these offline probability targets using Equation 11. The adapter-conditioned model receives the same query and response prefix, with historical information supplied through the generated adapter. Only response positions contribute to distribution matching. Averaging first within each pair and then within each session

gives equal session weight despite variable response lengths. Reference support size, sequence limits, and optimizer settings are specified in Appendix E.4.

C.2 Cross-Session Consolidation Data

Training-example construction. Each downstream example comprises the chronological history available at its query boundary, the query, candidate answers, and the correct answer label. The history is converted to bounded inputs using the compilation-stage event representation. Native session boundaries are preserved where available; long sessions are divided into chronological segments with a 4,096-token budget and one-event overlap. Oversized events are split into token chunks to retain their content. The session encoder consumes the history segments, while the query and candidate answers are supplied to the answering backbone. The correct label supplies task supervision. Frozen session encodings are cached with their order and segmentation metadata so that multiple questions can reuse the same encoded history.

PERMA. For every evaluated history variant, each question is paired with its query-specific visible session prefix and official A–H candidate set. Ten leave-one-user-out folds assign nine users to gate training and the remaining user to evaluation. All three task types of the training users contribute supervision. Each example starts a fresh recurrence over its visible history, with the first segment initializing memory and subsequent segments processed in chronological order. The held-out user's history is encoded at evaluation using the fixed compiler and trained gate.

PersonaMem-v2. Construction starts from the pinned 32K-Text release and its training, validation, and benchmark partitions. Training and validation rows belonging to benchmark personas are excluded. Two known invalid training rows are removed: one has no distractor answers and one includes the correct answer among the distractors. Null distractor entries are discarded with the repair recorded. The resulting partitions contain 18,527 training examples, 2,059 validation examples, and all 5,000 benchmark questions. Each example resolves its linked chat history into a chronological message stream, which is segmented under the shared token budget. Candidate answers are deterministically shuffled using the instance identifier, and the same ordering is used by every method. Training uses the training partition; the final-epoch gate is evaluated on the benchmark partition.

PrefEval. Each base example is represented in three evidence forms: an explicit statement, a choice-based interaction, and a persona-based interaction. The corresponding evidence sessions are followed by the official intervening dialogue prefix for each of the 10-, 70-, and 300-turn settings. The frozen preparation contains 1,000 base examples and 3,000 form-specific examples before expansion over these three history lengths. The topic split follows the official training implementation with seed 42: 16 topics are used for training, while transportation, technology shopping, education resources, and motor shopping are held out. One gate is trained jointly across training topics, evidence forms, and history lengths. The split contains 820 training and 180 held-out base examples, yielding 2,460 and 540 form-specific examples, respectively. Expansion over the three history lengths produces 7,380 training instances and 1,620 evaluation instances; each history-length setting evaluates the same 540 held-out form-specific examples. Base examples and topics are disjoint across the training and evaluation partitions. Shared intervening sessions reuse their cached encodings. Answer options are deterministically permuted with seed 42 and the instance identifier, consistently across methods.

Supervision and optimization. For these multiple-choice experiments, the task loss in Equation 14 supplies cross-entropy supervision on the correct answer label after option permutation. On PERMA, training uses the full-vocabulary next-token logits with the correct answer-label token as the target; evaluation selects the highest-logit legal option. Gradients pass through the frozen answering backbone and decoder to the recurrent gate. Across training examples, this supervision learns a shared integration policy for the application's memory requirements. The policy is reused across subsequent interactions with its parameters fixed after deployment. Training parameters and checkpoint selection are specified in Appendix E.4; evaluation metrics and aggregation are specified in Appendix E.1.

C.3 Query–Response Generation Prompts

The following prompt examples illustrate grounded query–response generation from session histories. Wording and output-format notation are simplified for readability while preserving the generation requirements.

Query–Response Generation: System Prompt

Create concise question–answer pairs grounded in the provided agent session.

Requirements

1. Use only facts explicitly present in the session history.
2. For each answerable question, provide a reference answer and cite one or more supporting events. Copy event identifiers exactly from the history; do not invent identifiers or cite the session as a whole.
3. Mark unanswerable questions accordingly and leave their supporting event list empty.
4. Keep questions and reference answers concise.

Output

Return one compact JSON object containing each question, its type, answerability, reference answer, and supporting event identifiers. Do not include additional prose, Markdown, memory summaries, or reasoning.

Query–Response Generation: User Prompt

Generate the remaining grounded question–answer pairs so that the session has exactly ten pairs in total. Do not repeat any existing questions.

Existing questions, if any: [Existing questions]

Session history: [Session events with their identifiers]

D Ablation Details

D.1 Compilation Objective Variants

The objective ablation in Table D.1 varies two properties of compiler training: the response trajectory used to evaluate the token distributions and the divergence defined on those distributions. We use the notation introduced in subsection 2.2. For pair i and response position j, $p_{i,j}^{\mathrm{ctx}}$ is the next-token distribution produced when the frozen language model receives the session as textual context, and $p_{i,j}^{\mathrm{mem}}$ is the distribution produced when the same model receives the compiler-generated adapter. The fixed response token at this position is $y_{i,j}$, and $S_{i,j}$ contains the K highest-probability tokens under $p_{i,j}^{\mathrm{ctx}}$.

Fixed-trajectory objectives. Compiler SFT treats each fixed reference response as a sequence of hard token targets. Its token-level loss is

$$\ell_{i,j}^{\mathrm{SFT}} = -\log p_{i,j}^{\mathrm{mem}}(y_{i,j}). \tag{D.1}$$

Here, $\ell_{i,j}^{\mathrm{SFT}}$ is the negative log-likelihood of reference token $y_{i,j}$ under the adapter-conditioned distribution. Top-K CE replaces the hard target with the probabilities of the selected context-conditioned tokens:

$$\ell_{i,j}^{\mathrm{TopK}} = -\sum_{v \in S_{i,j}} p_{i,j}^{\mathrm{ctx}}(v) \log p_{i,j}^{\mathrm{mem}}(v). \tag{D.2}$$

The symbol v indexes a token in the shared vocabulary, and $\ell_{i,j}^{\mathrm{TopK}}$ denotes the selected-support cross-entropy. Fixed FKL uses the same fixed response trajectory and selected support, then represents all tokens outside $S_{i,j}$ by their aggregate probability mass. Its token-level loss is defined in

Equation 9, and the session-balanced aggregation and adapter regularization follow Equation 10 and Equation 11.

Sampled-trajectory objectives. The sampled-trajectory variants sample a response $\widetilde{y}_i = (\widetilde{y}_{i,1}, \ldots, \widetilde{y}_{i,\widetilde{N}_i})$ from the adapter-conditioned model, where $\widetilde{N}_i$ is its generated length and $\widetilde{y}_{i,<j}$ is the prefix preceding position j. Both models are then evaluated along this shared sampled prefix:

$$\widetilde{p}^{\mathrm{ctx}}_{i,j}(v) = p_\theta(v \mid c, x_i, \widetilde{y}_{i,<j}). \tag{D.3}$$

$$\widetilde{p}^{\mathrm{mem}}_{i,j}(v) = p_{\theta \oplus \Lambda(c)}(v \mid x_i, \widetilde{y}_{i,<j}). \tag{D.4}$$

The tilde distinguishes distributions evaluated on the sampled trajectory from the fixed-trajectory distributions in Equation 6 and Equation 7. Sampled FKL applies the forward-KL construction in Equation 9 to $\widetilde{p}^{\mathrm{ctx}}_{i,j}$ and $\widetilde{p}^{\mathrm{mem}}_{i,j}$.

Sampled RKL selects the top-K support $\widetilde{S}_{i,j}$ under $\widetilde{p}^{\mathrm{mem}}_{i,j}$ and reverses the order of the two distributions. For $a \in \{\mathrm{ctx}, \mathrm{mem}\}$, let $\widetilde{\tau}^{a}_{i,j}$ denote the probability mass that $\widetilde{p}^{a}_{i,j}$ assigns outside $\widetilde{S}_{i,j}$. Its token-level objective is

$$\ell^{\mathrm{RKL}}_{i,j} = \sum_{v \in \widetilde{S}_{i,j}} \widetilde{p}^{\mathrm{mem}}_{i,j}(v) \log \frac{\widetilde{p}^{\mathrm{mem}}_{i,j}(v)}{\widetilde{p}^{\mathrm{ctx}}_{i,j}(v)} + \widetilde{\tau}^{\mathrm{mem}}_{i,j} \log \frac{\widetilde{\tau}^{\mathrm{mem}}_{i,j}}{\widetilde{\tau}^{\mathrm{ctx}}_{i,j}}. \tag{D.5}$$

Here, $\ell^{\mathrm{RKL}}_{i,j}$ is the top-K-plus-tail approximation to the reverse KL at the sampled response position.

Sampled RKL (K3) estimates the same divergence from the sampled response token. We define the sampled-token log-ratio as

$$r_{i,j} = \log \widetilde{p}^{\mathrm{ctx}}_{i,j}(\widetilde{y}_{i,j}) - \log \widetilde{p}^{\mathrm{mem}}_{i,j}(\widetilde{y}_{i,j}). \tag{D.6}$$

The corresponding K3 loss is

$$\ell^{\mathrm{K3}}_{i,j} = \exp(r_{i,j}) - r_{i,j} - 1, \tag{D.7}$$

where $r_{i,j}$ compares the two models' log-probabilities for sampled token $\widetilde{y}_{i,j}$ and $\ell^{\mathrm{K3}}_{i,j}$ is its nonnegative single-sample reverse-KL estimator. All variants average token losses within pairs and then across pairs, and all use the same adapter regularization as Equation 11. Each sampled-trajectory run begins with 10,377 SFT warm-start updates, then switches to its respective objective for the remaining 41,508 updates within the same run, reaching the common 51,885-update budget. Every trained compiler is frozen before fitting an independent consolidation gate with the same downstream architecture and optimization protocol.

Table D.1: Effect of compiler training objectives on PERMA accuracy (%).

Objective	Clean SD	Noisy SD	Clean MD	Noisy MD	Avg.
Compiler SFT	72.43	71.97	50.47	49.98	61.21
Sampled RKL (top-K)	73.53	74.63	60.49	58.97	66.91
Sampled RKL (K3)	71.03	70.60	63.42	63.02	67.02
Top-K CE	82.10	83.87	73.30	71.37	77.66
Sampled FKL	88.90	87.83	79.23	79.04	83.75
Fixed FKL	**90.57**	**91.20**	**79.67**	**80.62**	**85.52**

D.2 Compilation Training Dynamics

We evaluate intermediate compilation checkpoints by freezing each compiler and fitting an independent consolidation gate with the same downstream protocol. Figure D.1 reports both the average trajectory over all seven PERMA settings and the corresponding trajectory within each setting. Fixed FKL establishes the highest average downstream accuracy by approximately 15,000 updates and preserves this lead through the 51,885-update budget. The same ordering appears throughout the seven individual settings, while Compiler SFT exhibits an early decrease followed by a gradual recovery. These trajectories show that the endpoint comparison in Table D.1 reflects a stable difference in compilation quality over training.

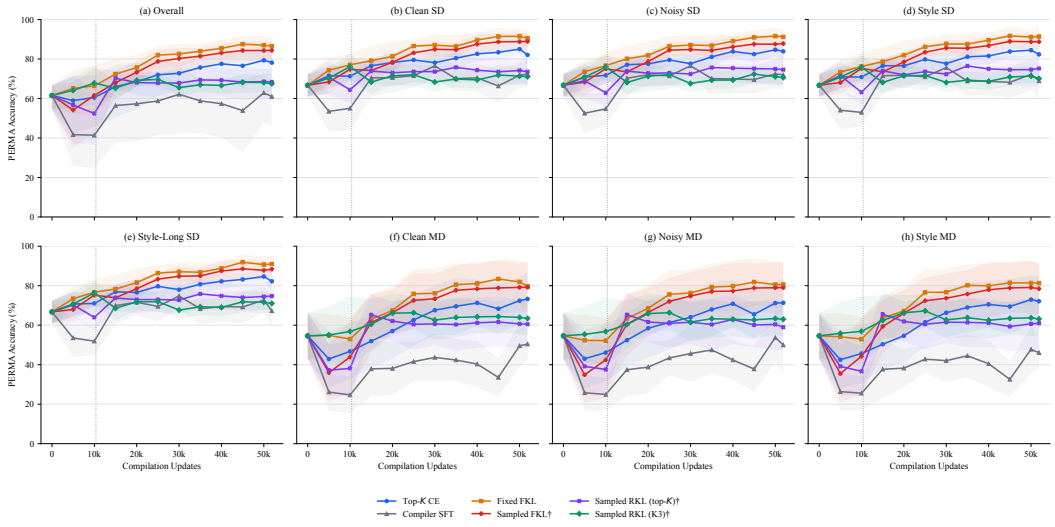


Figure D.1: Compilation training dynamics on PERMA. Panel (a) reports the unweighted average over all seven settings; its shaded regions span the minimum and maximum setting-level accuracy. Panels (b)–(h) report each setting separately; their shaded regions show the standard deviation across the ten held-out-user folds. Every checkpoint is evaluated after independently fitting the same consolidation gate architecture. Daggered variants switch to their respective objectives after 10,377 SFT warm-start updates within each run.

D.3 Cross-Session Consolidation Variants

The consolidation ablation in Table D.2 fixes the Fixed FKL compiler and changes the operation applied to the chronologically ordered sequence of memory representations. All variants share the compiler checkpoint, input segmentation, visible-history boundary, held-out-user folds, and evaluation protocol. The compiler and backbone remain frozen; the learned rule fits the consolidation gate on the training users of each fold.

Factor convention. Let S denote the number of compiler segments in the visible history and $s \in \{1, \ldots, S\}$ index their chronological order. For one adapted layer and module, suppressing the indices ℓ, m, let $A_s \in \mathbb{R}^{r \times d^{\text{in}}}$ and $B_s \in \mathbb{R}^{d^{\text{out}} \times r}$ be the decoded factors of segment s. Here r is the generated factor width, and $d^{\text{in}}, d^{\text{out}}$ are the target module's input and output dimensions. The decoder also supplies a shared factor pair A_0, B_0, called the head bias in the implementation, which contributes the history-independent update $\Delta W_0 = \gamma B_0 A_0$. The coefficient γ is the checkpoint's LoRA scaling from Equation 5. With head bias enabled, this shared pair occupies one additional rank-r block.

Selection and latent averaging. Latest Session retains all compiler segments belonging to the final observed semantic session and combines their decoded factors by rank concatenation. Latent averaging takes the element-wise mean of all S segment representations and decodes that single mean. Learned consolidation instead decodes the accumulated memory produced by subsection 2.3. Both latent averaging and learned consolidation produce one rank-r generated block plus the shared bias block.

Rank concatenation. Concatenation stacks the down-projection factors vertically and the up-projection factors horizontally, appending the shared bias pair once:

$$A_{\text{cat}} = [A_1^\top, \ldots, A_S^\top, A_0^\top]^\top, \tag{D.8}$$

$$B_{\text{cat}} = [B_1, \ldots, B_S, B_0], \tag{D.9}$$

$$\Delta W_{\text{cat}} = \gamma B_{\text{cat}} A_{\text{cat}} = \gamma \sum_{s=1}^{S} B_s A_s + \Delta W_0. \tag{D.10}$$

The resulting factor width is $(S+1)r$ with head bias enabled; Latest Session uses the same construction over the selected session's segments. The update is an unnormalized sum under the unchanged coefficient γ.

LoRA factor averaging. This rule computes the arithmetic mean of each generated factor separately:

$$\overline{A} = \frac{1}{S}\sum_{s=1}^{S} A_s, \tag{D.11}$$

$$\overline{B} = \frac{1}{S}\sum_{s=1}^{S} B_s, \tag{D.12}$$

$$\Delta W_{\text{fac}} = \gamma \overline{B}\,\overline{A} + \Delta W_0 = \frac{\gamma}{S^2}\sum_{s=1}^{S}\sum_{u=1}^{S} B_s A_u + \Delta W_0. \tag{D.13}$$

Here u independently indexes the down-projection factor in the expanded product. The shared bias factors are identical across segments and remain unchanged by averaging, giving a total factor width of $2r$. Thus this operator includes cross-segment products $B_s A_u$. Averaging the induced updates is a distinct operator, $\gamma S^{-1}\sum_s B_s A_s + \Delta W_0$; the reported row evaluates factor averaging.

The variants in Table D.2 retain the same checkpoint scaling γ, applied directly to the factor product without division by the assembled factor width. Update norms follow from the specified operators, with no additional post-aggregation norm calibration. Concatenation has history-dependent factor width, whereas latent averaging, factor averaging, and learned consolidation have fixed factor width.

Table D.2: Cross-session consolidation ablation with the Fixed FKL compiler on PERMA (accuracy, %).

Consolidation	Clean SD	Noisy SD	Clean MD	Noisy MD	Avg.
Rank concatenation	14.83	15.10	12.70	12.71	13.84
Latent averaging	63.13	61.67	28.67	28.61	45.52
Latest Session	68.07	67.37	38.96	39.83	53.56
Factor averaging	69.50	69.37	38.99	38.99	54.21
Learned gate (ours)	**90.57**	**91.20**	**79.67**	**80.62**	**85.52**

Additional consolidation experiments.

Experimental setup. We evaluate alternative consolidation rules on PERMA Clean SD and Clean MD using Qwen3-8B and the same frozen five-pass memory compiler. Each setting uses ten held-out-user folds, with nine users available for training or coefficient selection and the remaining user reserved for evaluation. All methods share the compiler, decoder, backbone, history segmentation, and evaluation questions. Histories are segmented at 4,096 tokens with one-message overlap; updates operate on these compiler segments. Latent averaging and RPMem are rerun alongside the additional controls. Accuracy is macro-averaged over the ten held-out users, and the two-setting mean gives equal weight to Clean SD and Clean MD.

Aggregation rules. Latent averaging takes the arithmetic mean of the segment representations before decoding. Tuned exponential moving averaging (EMA) initializes $h_1 = q_1$ and applies $h_t = \alpha h_{t-1} + (1-\alpha)q_t$ thereafter. For each fold and setting, we select α from $\{0, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99\}$ using macro-average accuracy on the nine training-side users, with ties resolved toward the smaller coefficient. The selected coefficient is fixed for the held-out user. Update averaging decodes each segment separately and averages its induced weight update:

$$\Delta W = \frac{\gamma}{S}\sum_{s=1}^{S} B_s A_s + \Delta W_0,$$

where S is the number of segments, γ is the compiler's LoRA scale, and ΔW_0 is the shared decoder bias update, included once. This rule averages complete updates rather than the two factors separately. Its factor-concatenation implementation retains a rank that grows with S.

Learned consolidation. The scalar gate mean-pools the previous memory and incoming representation over their layer, module, and rank axes, concatenates the resulting two 512-dimensional vectors, and applies a learned linear map and sigmoid. The resulting retention coefficient is shared across all memory coordinates; the gate has 1,025 parameters. RPMem instead uses coordinate-dependent retention. Both learned methods initialize $h_1 = q_1$ and apply consolidation from the second segment onward. Both are trained on the other nine users with the same answer-token cross-entropy objective for five epochs, using AdamW with learning rate 10^{-3}, weight decay 0.01, gradient clipping at 1.0, and seed 42. The compiler and backbone remain frozen, and neither gate receives the evaluation question as an input.

Table D.3: Additional consolidation experiments on PERMA (accuracy, %). Each setting averages ten held-out-user folds. Mean averages the two displayed settings; it is distinct from the four-setting Core Avg. in the main results. All values are from the matched evaluation runs described here.

Method	Task supervision	Clean SD	Clean MD	Mean
Latent averaging	None	63.13	28.67	45.90
Update averaging	None	68.57	38.12	53.34
Scalar gate	Answer-token training	67.57	39.21	53.39
Tuned EMA	Coefficient selection	68.07	38.96	53.51
RPMem	Answer-token training	**90.80**	**79.13**	**84.97**

Results. Update averaging and tuned EMA achieve two-setting means of 53.34% and 53.51%, respectively, compared with 45.90% for latent averaging. EMA selects $\alpha = 0$ in all twenty setting–fold combinations, favoring the most recent segment within the evaluated coefficient family. The learned scalar gate reaches 53.39%, whereas RPMem reaches 84.97%. Under the same task supervision and training budget, coordinate-dependent retention therefore substantially outperforms a single shared retention coefficient on both settings. These comparisons support the value of fine-grained recurrent consolidation beyond uniform averaging and globally weighted updates.

D.4 Amount of Consolidation Supervision

We vary the number of training users from one to nine and evaluate each gate on users excluded from its training set. For every training-set size, cyclic sampling produces ten balanced train-evaluation partitions, and results are first aggregated within users. As shown in Figure D.2, one-user adaptation reaches 79.84% on Clean SD, and five-user adaptation reaches 89.98%. Accuracy remains between 89.98% and 91.47% from five to nine users. The learning curve demonstrates that the consolidation policy transfers across users with limited downstream supervision.

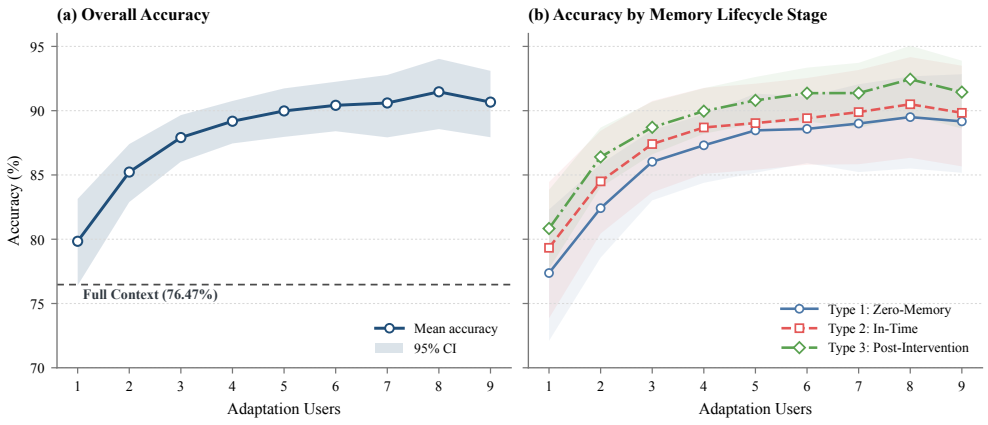


Figure D.2: Data efficiency of consolidation adaptation on PERMA Clean SD. Curves report held-out users, and shaded regions show user-level bootstrap 95% confidence intervals. The horizontal reference marks Full Context at 76.47%.

E Full Experimental Configuration and Hyperparameters

E.1 Evaluation Scope and Benchmark Protocols

PERMA. PERMA contains ten simulated users with temporally ordered interaction histories. Preferences are introduced, extended, or revised by events in these sessions, and each evaluation query presents eight candidate answers labeled A through H. The benchmark crosses two history structures, single-domain (SD) and multi-domain (MD), with clean and noisy history realizations to form four core settings. Style SD and Style MD rewrite the histories in user-specific linguistic styles, while Style-Long SD adds realistic long-dialogue distractors to the rewritten single-domain history. Core Avg. is the unweighted mean of the four core settings. The seven-setting average is used only in analyses that explicitly report all seven settings. Task-adapted methods use ten-fold leave-one-user-out evaluation: every fold trains on nine users and reports predictions for the remaining user, after which accuracy is averaged over the ten held-out users.

PersonaMem-v2. We use the frozen 32K-Text release, comprising 18,527 training examples, 2,059 validation examples, and 5,000 independent benchmark questions. Each example associates a long user history with a multiple-choice query. The annotations identify whether the queried information concerns the focal user or another person and whether the relevant state is current, updated, or marked for forgetting. Overall is the official aggregate accuracy. Self restricts evaluation to information about the focal user, and Current restricts it to the currently valid state. These two slices accompany Overall in the compact main table because they directly expose the user-specific state maintained across the history. The complete ownership and update-status breakdown is retained in the benchmark result artifacts.

PrefEval. The evaluation uses the official split of 16 training topics and four held-out topics. It covers explicit statements, implicit choice-based evidence, and implicit persona-driven evidence, each evaluated after 10, 70, and 300 turns of intervening dialogue. For each history length, the main table reports the unweighted mean accuracy over the three evidence forms. Topic-disjoint evaluation measures whether the downstream consolidation policy transfers to semantic domains absent from its training partition.

E.2 Baseline Implementations

Reference conditions. No Context presents the query and candidate answers without historical information. Gold State converts the state annotation supplied by a benchmark into structured text and places it before the query. The latter condition measures answer selection given direct access to the annotated state; its result also reflects the coverage and granularity of the benchmark annotation.

Textual-memory methods. Full Context concatenates all visible sessions in chronological order and places the resulting history before the query. RAG divides the same visible history into retrievable units, embeds them with M3-Embedding (BGE-M3), and injects the ten units with the highest dense-retrieval scores. On PERMA, units are non-overlapping groups of two consecutive messages from the chronologically flattened history, including role labels. Retrieved units are presented in descending relevance order; embedding inputs are truncated at 8,192 tokens. Rolling Summary updates a 1,024-token summary after every session by combining the previous summary with the incoming interaction. The PERMA Mem0 run uses the open-source v2.0.12 pipeline, Qwen3-8B for memory extraction, and BGE-M3 for retrieval. LightMem applies its hierarchical semantic compression and retrieval pipeline to the same visible history. Every method receives an identical query and candidate-answer interface after constructing its textual memory input.

SFT. This trained control freezes Qwen3-8B and fits a rank-8 LoRA with scale 32 on the `down_proj` modules. For each PERMA variant and held-out user, the adapter is trained for five epochs on the raw history sessions of the other nine users with the standard assistant-response objective. The resulting 70 adapters follow the same leave-one-user-out boundary as downstream consolidation. Training supervision consists of dialogue responses from the training histories, while evaluation supplies the held-out user's complete visible history before each query. The same construction follows the official training partition on the other main benchmarks.

Metis. We load the released Metis-9B memory checkpoint with its native Qwen3.5-9B backbone. Historical sessions are written sequentially through the model's native recurrent memory mechanism, preserving their original order. The memory state is reset at the beginning of every independent benchmark task, and the released checkpoint is evaluated directly through the shared non-thinking answer-selection interface. Results for the released 4B, 9B, and 27B checkpoints are reported in Appendix H.2; the main table uses the strongest overall scale, Metis-9B.

E.3 Shared Evaluation and Training Protocol

In the main comparison, RPMem and our baseline implementations use Qwen3-8B as the answering model. We directly evaluate the released Metis-9B model. Generalization and transfer use the five backbones reported in Appendix E.7 and Appendix G. Thinking mode is disabled throughout. Within each benchmark, methods receive the same chronological history boundary, query, candidate answers, and output parser. On PERMA, prediction selects the largest next-token logit among the legal labels A through H. Accuracy is first computed for each held-out user and then macro-averaged across the ten folds. PersonaMem-v2 uses its frozen training, validation, and benchmark splits, and PrefEval uses its official topic-disjoint split.

The default RPMem run losslessly segments sessions at 4,096 tokens and overlaps one complete event between adjacent segments. Its compiler is the 51,885-update checkpoint trained with the forward-KL objective in Equation 11. Each stored reference position contains the 32 highest-probability context-conditioned tokens and a tail category containing the remaining probability mass. Each accepted compilation session contributes eight training pairs and two disjoint query-level validation pairs. During downstream adaptation, the context encoder, Perceiver-based resampler, LoRA decoder, and answering model remain fixed while the consolidation gate is optimized on the training partition. Table E.1 records the default PERMA gate configuration; compiler architecture, optimization, and the remaining downstream protocols appear in Appendix E.4.

Table E.1: Primary Cross-Session Memory Consolidation configuration used for PERMA.

Component	Setting
Backbone	Qwen3-8B, non-thinking mode
Data split	10-fold leave-one-user-out
Compiler	Fixed FKL checkpoint 51,885
Maximum segment length	4,096 tokens
Adjacent-segment overlap	One event
Trainable module	Consolidation gate only
Trainable parameters	524,800
Initial memory	$h_0 = 0$; $h_1 = q_1$
Gate initialization	Weight 0, bias -2.0
Optimizer	AdamW
Learning rate	10^{-3}
Weight decay	0.01
Gradient clipping	1.0
Training epochs	5
Batch size	1
Random seed	42
Prediction	Maximum A–H next-token logit

E.4 Compiler Architecture and Optimization

Architecture. The primary Qwen3-8B compiler uses a frozen ModernBERT-base context encoder. Its hidden states are mapped to the 36 target-layer positions by uniformly spaced encoder indices, with repeated indices when the target has more layers than the encoder. LoRA is applied to the feed-forward down-projection (`down_proj`) in each target layer, giving $L = 36$ and $M = 1$. With $r = 8$ and $d = 512$, each session memory has shape $36 \times 1 \times 8 \times 512$. Table E.2 specifies the encoder and decoder.

Each decoder residual block applies layer normalization, a layer-specific linear expansion, SiLU, a linear contraction, and a second layer normalization before adding the residual. After the four

Table E.2: Primary Qwen3-8B compiler architecture.

Component	Configuration
Frozen models	Qwen3-8B; ModernBERT-base
Adapted layers / module types	$L = 36$; $M = 1$, `down_proj`
Generated factor width / latent width	$r = 8$; $d = 512$
Perceiver latent queries	8
Perceiver encoder / output blocks	9 / 1 cross-attention blocks
Attention heads / head width	8 / 64
Perceiver feed-forward width	2,048
Additional self-attention blocks	0
Decoder pre-head	4 layer-specific residual MLP blocks
Decoder MLP widths	$512 \to 2{,}048 \to 512$
Decoder output	Layer-specific linear factor projections
Shared head-bias block	Enabled; width 8
LoRA scaling / dropout	$\gamma = 32$ / 0

blocks, each latent vector is normalized by its Euclidean norm and projected to the concatenated input- and output-factor dimensions. Splitting this projection yields the two factors, each multiplied by a learned layer- and rank-specific scalar. These scalars are initialized to one for A and zero for B. For the shared head-bias block, entries of A_0 are initialized from a zero-mean Gaussian with standard deviation $0.2/\sqrt{rd^{\text{in}}}$, while B_0 is initialized to zero. This block is appended once during adapter assembly, as defined in Appendix D.3. The coefficient γ directly multiplies the factor product, using the same scaling convention across assembled factor widths.

Factor regularization. For a bounded training session c, the implementation first computes the mean absolute value of each generated factor, adds the A and B terms, and averages over layers and module types:

$$\mathcal{R}_1(c) = \frac{1}{LM} \sum_{\ell=1}^{L} \sum_{m=1}^{M} \left(\frac{\|A_{\ell,m}(c)\|_1}{rd^{\text{in}}_{\ell,m}} + \frac{\|B_{\ell,m}(c)\|_1}{rd^{\text{out}}_{\ell,m}} \right). \tag{E.1}$$

Here, $\|\cdot\|_1$ denotes the sum of absolute entries, and $A_{\ell,m}(c), B_{\ell,m}(c)$ are the generated factors for session c at layer ℓ and module type m after their learned scalar multipliers. The denominators are the respective entry counts, using the dimensions defined in subsection 2.2. Both factors receive equal weight despite different input and output widths. The penalty is evaluated before shared-bias concatenation and application of γ, and uses $\lambda = 0.01$ in Equation 11.

Compilation optimization. Training updates the Perceiver-based resampler and decoder while freezing both language models. Each of the 664,128 training sessions contributes eight pairs; response-token losses are averaged within pairs and then within sessions. The recorded global batch contains 64 sessions, obtained from eight workers, one session per worker, and eight gradient-accumulation steps. Five passes therefore give $664{,}128 \times 5/64 = 51{,}885$ optimizer updates. The final completed checkpoint is used for downstream evaluation. Table E.3 records the primary training configuration.

Downstream optimization. The PERMA settings appear in Table E.1. For PersonaMem-v2 and PrefEval, the formal training protocol uses the shared configuration in Table E.4. Each training question produces one optimizer update. PersonaMem-v2 shuffles personas and then their training questions using the fixed seed; the chronological order of the history within each question is preserved. PrefEval trains one gate jointly across the 16 training topics, three information forms, and three primary history lengths, and evaluates on the four held-out topics. The data construction and split definitions appear in Appendix C.2. In all cases, optimization updates only the consolidation gate; the compiler and answering backbone remain frozen. The final-epoch gate is used for evaluation.

E.5 Training and Query-Cost Details

Streaming deployment protocol. We profile accumulated source-session counts of $1, 2, 4, 8, 16, 32,$ and 64 on eligible PERMA Clean-SD histories. All models and services are

Table E.3: Primary fixed-reference forward-KL compilation configuration.

Component	Setting
Training sessions / pairs per session	664,128 / 8
Retained reference support	$K = 32$, with aggregate tail mass
Maximum session length	4,096 tokens
Maximum query-response sequence	640 tokens
Maximum reference-model sequence	4,864 tokens
Global batch / accumulation steps	64 sessions / 8
Workers / sessions per worker	8 / 1
Training passes / updates	5 / 51,885
Optimizer	AdamW, $\beta_1 = 0.9$, $\beta_2 = 0.999$
Configured learning rate	4×10^{-5}
Warmup updates	500
Weight decay / gradient clipping	0.01 / 1.0
Factor-regularization weight	$\lambda = 0.01$
Random seed	42

Table E.4: Shared downstream training configuration for PersonaMem-v2 and PrefEval.

Setting	Value
Trainable module	Recurrent consolidation gate
Initial memory	$h_0 = 0$; $h_1 = q_1$
Gate weight / bias initialization	$W_g = 0$; $b_g = -2$
Training objective	Cross-entropy over legal answer labels
Optimizer / learning rate	AdamW / 10^{-3}
Weight decay / gradient clipping	0.01 / 1.0
Training epochs / random seed	5 / 42
Questions per optimizer update	1
Evaluation checkpoint	Final epoch

resident before timing. A hot RPMem update includes segmentation and encoding of the newly arrived session, recurrent memory consolidation, and LoRA materialization. For adapter rank concatenation, we concatenate the accumulated history, repartition it into chunks of at most 4,096 tokens, and recompile all chunks at each update. Unlike the session-wise segmentation used in the consolidation ablation (Appendix D.3), these chunks can span session boundaries. The Mem0 timer includes LLM-based memory extraction, embedding, and vector-index updates, while Full Context has no separate write operation. Persistent memory counts per-user data retained for answering queries and processing subsequent updates, excluding shared model parameters such as the compiler and consolidation gate. Historical query tokens count the memory tokens supplied to the answering model. Growth values are the empirical ratios between the 64-session and one-session endpoints rather than asymptotic complexity claims.

General-capability retention. We evaluate the unmodified Qwen3-8B backbone (Base LLM, with no user-memory adapter) and ten user-specific memory states on MMLU (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), and IFEval (Zhou et al., 2023). Table E.5 reports the user-level mean and standard deviation for both memory methods.

Table E.5: General-benchmark performance (%) of the base model and memory-adapted models.

Method	MMLU (5-shot)	GSM8K (5-shot)	IFEval (0-shot)
Base LLM	71.96	75.21	81.70
Latest Session	69.85 ± 0.51	73.15 ± 1.63	77.38 ± 1.20
RPMem (ours)	70.13 ± 0.41	73.78 ± 1.53	77.89 ± 1.42

Query cost on PersonaMem-v2. The deployment measurements in the main text are complemented by an independent evaluation over the 5,000-question PersonaMem-v2 test set. Table E.6

reports the mean input length, language-model forward time, and parametric state associated with each method.

Table E.6: Mean query cost on the PersonaMem-v2 test set.

Method	Input tokens	Forward time (s)	Parametric state
Full Context	33,363.71	4.287	–
RAG (BGE-M3)	2,757.58	0.253	–
Latest Session	337.49	0.056	36 MiB, rank 16
Adapter rank concatenation	337.49	0.056	195.9 MiB, rank 87.1 mean
RPMem	**337.49**	**0.055**	**36 MiB, rank 16**

The rank reported in Table E.6 denotes the assembled factor width: eight generated components plus the eight-component shared head-bias block for RPMem and Latest Session (Appendix D.3), not the numerical rank of the update matrix. With an approximately 32K-token user history, RPMem reduces the mean forward time from 4.287 seconds under Full Context to 0.055 seconds while preserving a fixed-rank 36 MiB state. Rank concatenation receives the same query tokens but expands the adapter state with the number of accumulated sessions. At the 64-session endpoint of the deployment experiment, RPMem incurs 0.575 GiB of peak incremental GPU memory, compared with 2.826 GiB for adapter rank concatenation; this measurement is reported here because peak allocation is not available for the textual-memory pipelines.

Consolidation adaptation cost. For a representative Clean-SD fold with 630 training examples and 3,150 updates, the 524,800-parameter consolidation gate trains in 13.0 minutes on one NVIDIA A800-80GB and produces a 2.004 MiB checkpoint. The peak GPU-memory increase over the full training run is 12.173 GiB, measured relative to the resident-model baseline before gate training. This increase primarily reflects activations needed to backpropagate through the frozen decoder and language model rather than the gate parameters.

Table E.7: Consolidation-stage adaptation cost on a representative PERMA fold.

Measurement	Value
Trainable parameters	524,800
Training examples	630
Parameter updates	3,150 (5 epochs)
Training time	13.0 min
Peak incremental memory	12.173 GiB
Saved checkpoint size	2.004 MiB

To measure how adaptation scales with the memory sequence, we select ten Type-3 tasks containing at least 64 cached session representations, truncate each sequence to progressively longer prefixes, and repeat every measurement three times. Peak Δ reports the mean per-step peak memory increase above a warmed-up baseline with the model, gate, optimizer, and query tensors already resident. As shown in Table E.8, increasing the sequence from one to 64 representations raises total step time by $1.39\times$ and peak incremental memory by 3.6%.

Table E.8: Consolidation optimization cost as the number of cached session representations increases.

Sessions	Load (ms)	Compute (ms)	Total (ms)	Peak Δ (GiB)
1	0.8	159.0	159.8	5.059
2	1.4	161.1	162.5	5.062
4	2.6	161.6	164.3	5.069
8	4.7	161.2	165.9	5.081
16	8.3	164.1	172.4	5.107
32	17.3	169.6	186.9	5.153
64	43.1	179.6	222.7	5.242

Of the additional 62.9 ms at 64 sessions, 42.3 ms is spent loading cached representations and 20.6 ms in GPU computation. The measured scaling therefore localizes the principal growth term to cache I/O, while the recurrent consolidation computation and activation memory remain comparatively stable.

Compilation optimization cost. Table E.9 reports measured step times and estimated allocated GPU-hours for the objectives evaluated in Table D.1. A fixed-reference full-vocabulary reverse-KL run (Fixed RKL) is included as an additional runtime comparison. The three 8-GPU configurations have nearly identical step times and an estimated cost of 338–339 GPU-hours for 51,885 updates. Each of the four 24-GPU runs, including Fixed RKL, performs 10,377 SFT warm-start updates followed by 41,508 updates with its respective objective. The total estimate uses separately measured preflight step times for the two phases:

$$\text{GPU-hours} = \frac{24}{3600}\left(10{,}377\, t_{\text{SFT}} + 41{,}508\, t_{\text{objective}}\right).$$

Here, both step times are in seconds; only $t_{\text{objective}}$ is displayed for these runs in the table.

Table E.9: Compilation-stage optimization cost by training objective.

Objective	Prefix	Loss	Step (s)	Allocated GPUs	Est. GPU-hours
Top-K CE	Reference	Top-K CE	2.943	8	339
Fixed FKL	Reference	FKL	**2.932**	8	**338**
Compiler SFT	Reference	CE	2.940	8	339
Sampled RKL (K3)	Student	RKL (K3)	13.061	24	3,849
Fixed RKL	Reference	RKL	9.585	24	2,939
Sampled FKL	Student	FKL	18.202	24	5,284
Sampled RKL (top-K)	Student	RKL	15.944	24	4,627

The 24-GPU runs reserve 16 actor and eight teacher GPUs throughout training. Objective-stage timing includes online teacher scoring and, for sampled variants, student generation. Estimates exclude non-update overhead and separate offline reference preparation. Fixed FKL combines the highest downstream accuracy in Table D.1 with the lowest estimated allocated training cost among the evaluated distribution-matching configurations.

E.6 Complete Historical-Context Shift Results

Table E.10 reports the complete method matrix for the three historical-context shift settings summarized in subsection 3.2. Style SD and Style MD rewrite the visible histories in user-specific linguistic styles. Style-Long SD additionally inserts realistic long-dialogue distractors into the rewritten single-domain history. Avg. is the unweighted mean of the three settings.

Table E.10: Complete PERMA historical-context shift results (accuracy, %).

Method	Style SD	Style-Long SD	Style MD	Avg.
Full Context	76.40	75.83	71.35	74.53
Gold State	73.50	73.50	63.80	70.27
RAG (BGE-M3)	65.53	65.53	46.47	59.18
Rolling Summary	39.63	39.93	26.66	35.41
Mem0	58.73	59.47	40.35	52.85
LightMem	55.13	56.17	39.34	50.21
SFT	76.73	77.40	56.68	70.27
Metis-9B	<u>79.67</u>	<u>79.97</u>	<u>79.87</u>	<u>79.84</u>
RPMem (ours)	**91.37**	**90.97**	**81.33**	**87.89**

RPMem leads all three context-shift variants and reaches an average accuracy of 87.89%, exceeding Metis-9B and Full Context by 8.05 and 13.36 percentage points (pp), respectively. Relative to the corresponding clean settings, its accuracy changes by $+0.80$ pp on Style SD, $+0.40$ pp on Style-Long SD, and $+1.66$ pp on Style MD. Style-Long SD jointly varies linguistic style and history

length, so its result measures stability under the combined shift. Across all seven PERMA variants, RPMem reaches 86.53%, demonstrating that the parametric memory remains reliable as the style and length of the historical context change.

E.7 Complete Model-Generalization Results

Table E.11 reports the aggregate values underlying Figure 2. SD Avg. averages the four single-domain variants, MD Avg. averages the three multi-domain variants, and Overall Avg. averages all seven PERMA variants. Within each backbone, all methods use the same history boundary, non-thinking answer-selection protocol, and held-out-user folds. No Context results are aggregated from independent evaluations of each variant.

Table E.11: Complete aggregate results underlying the cross-model generalization comparison (accuracy, %).

Backbone	Method	SD Avg.	MD Avg.	Overall Avg.
Qwen3-4B	No Context	65.70	60.97	63.67
	Full Context	79.82	73.39	77.06
	RPMem (ours)	**85.98**	**79.04**	**83.01**
Qwen3-8B	No Context	66.73	54.60	61.53
	Full Context	75.43	70.67	73.39
	RPMem (ours)	**91.03**	**80.54**	**86.53**
Ministral-3-8B	No Context	69.57	74.47	71.67
	Full Context	77.05	77.56	77.27
	RPMem (ours)	**89.48**	**83.00**	**86.70**
Qwen3.5-9B	No Context	29.24	35.63	31.98
	Full Context	66.67	84.34	74.25
	RPMem (ours)	**96.21**	**92.59**	**94.66**
Qwen3.5-35B-A3B	No Context	76.00	84.11	79.48
	Full Context	57.72	74.38	64.86
	RPMem (ours)	**93.65**	**94.96**	**94.21**

F Memory Dynamics Analysis Details

Table F.1 reports the functional lifecycle comparison that motivates the update-dynamics analysis in the main text. T1 precedes the target memory event, T2 immediately follows it, and T3 follows subsequent intervening sessions. The table reports the memory-bearing T2 and T3 checkpoints averaged across all seven PERMA variants.

Table F.1: Memory-bearing accuracy (%) across the PERMA lifecycle.

Method	In-Time (T2)	Post-Intervention (T3)	T3–T2 (pp)
Full Context	76.3	77.2	$+0.9$
Latent averaging	46.5	45.6	-0.9
RPMem (ours)	**86.7**	**86.5**	-0.2

F.1 Source Decomposition of the Recurrent State

With the direct initialization $h_1 = q_1$, unrolling the recurrent update in equation 13 for $t \geq 2$ assigns an explicit coordinate-wise weight to every source session:

$$h_T = \sum_{i=1}^{T} w_{i,T} \odot q_i, \tag{F.1}$$

$$w_{1,T} = \prod_{k=2}^{T} z_k, \tag{F.2}$$

$$w_{i,T} = (1 - z_i) \odot \prod_{k=i+1}^{T} z_k, \qquad 2 \leq i \leq T. \tag{F.3}$$

Here, i indexes the source session, k indexes a subsequent update, and $w_{i,T}$ has the same shape as q_i. Its entries are the coefficients of that source in the accumulated memory after step T. All products are element-wise, and an empty product is the all-ones tensor. Thus $w_{1,1} = 1$, while $w_{i,i} = 1 - z_i$ for $i \geq 2$. The coefficients are nonnegative and sum to one at every coordinate. They describe the source decomposition along the observed gate trajectory; perturbing a session can also change later gates and therefore requires a new forward replay.

F.2 Trajectory Replay and Statistical Protocol

The analysis uses the Qwen3-8B compiler trained with fixed-reference forward KL. For each of the ten held-out PERMA users, trajectories are replayed with the consolidation module trained on the other nine users for the corresponding benchmark variant. Clean SD and Noisy SD each contain 423 task-conditioned trajectories and 34,218 session occurrences, giving 846 trajectories and 68,436 session occurrences in total. These counts include shared history appearing in different task-conditioned trajectories; the independent evaluation units are the ten users. Each semantic session in this replay occupies one compiler segment.

For matched comparisons, we first average the paired differences within each user and then average the ten user estimates with equal weight. The 95% confidence intervals are the 2.5th and 97.5th percentiles of 10,000 bootstrap means obtained by resampling users with replacement. Matching remains fixed during resampling. These intervals summarize variation across held-out users conditional on the fitted fold models.

Reading the main dynamics figure. In Figure 3(a–b), each colored point is one held-out user's paired-condition mean, and the hollow diamond averages the ten users. The diagonal marks equal write magnitude. In (c–d), violins show the matched-pair distributions, overlaid boxes span the interquartile range with median lines and 5th–95th percentile whiskers, and colored points show user means. Panel (e) reports each user's paired difference and its across-user mean; horizontal error bars give the user-bootstrap 95% confidence interval. The differences are Same minus Cross for write-pattern similarity and Cross minus Same for one-step survival, so positive values indicate stronger same-domain alignment and revision, respectively. In (f), the shaded bands give user-bootstrap 95% confidence intervals for the mean survival curves.

Event-role matching is performed without replacement within the same user, trajectory, and segment count. A minimum-total-cost assignment uses the sum of absolute differences in normalized session position and log token count, each scaled by its within-stratum interquartile range. A zero interquartile range is replaced by one. We report the resulting comparisons as associations along the observed histories and assess residual covariate balance using absolute standardized mean differences.

F.3 Session-Level Write Statistics

For a semantic session e containing one or more compiler segments, let Z_e denote the coordinate-wise retention of memory present before the event. For events following initialization, $Z_e = \prod_{k \in e} z_k$, where k indexes the event's compiler segments. The event containing the first segment has $Z_e = 0$ because that segment directly initializes memory. We summarize the resulting update

by the scalar write magnitude

$$w_e = 1 - \mathrm{mean}(Z_e). \tag{F.4}$$

The mean is taken over all layer, module, rank, and latent-feature coordinates of Z_e. Retention and write statistics are computed over fusion steps $t \geq 2$, following the direct initialization $h_1 = q_1$. Within each trajectory, the write range is the maximum minus the minimum write magnitude over these fusion steps; the reported trace range averages these ranges across trajectories. The untrained gate assigns $z = \sigma(-2) = 0.119$ to every coordinate of each gated update. Downstream training raises the mean retention to 0.670 and produces an average within-trajectory range of 0.141 in write magnitude, as shown in Table F.2.

Table F.2: Consolidation statistics on PERMA Clean SD for fusion steps $t \geq 2$.

Consolidation	Retain mean	Retain P5/P50/P95	Write mean	Trace range
Untrained gate	0.119	0.119/0.119/0.119	0.881	0.000
Learned gate	0.670	0.636/0.672/0.698	0.330	0.141

Domain-emergence and supplement events correspond to PERMA's preference-emergence and preference-supplement annotations, respectively. The semantic-role comparison uses 810 matched emergence–supplement pairs. The mean write difference is $+0.056$ with a user-level bootstrap 95% confidence interval of $[0.039, 0.073]$. After matching, the absolute standardized differences are 0.055 for log token count and 0.962 for normalized position. The association therefore reflects semantic role together with its temporal placement in the benchmark.

The Clean/Noisy comparison aligns sessions by user, task identifier and type, session index, date, event type, domain, and trajectory role. All 34,218 session occurrences align across the two variants; the reported perturbation comparison selects the 18,234 pairs whose conversation text differs. Noise changes text within the aligned histories while preserving their event structure. Each variant uses its corresponding trained gate, so this comparison measures the behavior of the complete trained recurrence under the two benchmark conditions. The mean Noisy-minus-Clean difference is $+0.011$, and the mean absolute per-session drift is 4.8% of the Clean write magnitude.

F.4 Domain Updates and Long-Term Survival

For the domain analysis, we flatten, center, and normalize the coordinate-wise write pattern $1 - Z_e$ and compute cosine similarity for matched same-domain and cross-domain event pairs. One-step survival is the ratio between a historical source coefficient immediately after and before a new update. Same-domain and cross-domain write-pattern similarities are 0.872 and 0.784, respectively; the corresponding one-step survival values are 0.458 and 0.818. Write-pattern pairs are matched within user and trajectory, with equal source and target segment counts. Matching covariates include both event positions, their temporal separation, and both log token counts. Transition comparisons similarly match source age, incoming-event position, and both log token counts. These comparisons characterize domain-associated update patterns in the observed trajectories.

At recurrence-step resolution, source i's survival after a subsequent updates is

$$L_i(a) = \frac{w_{i,i+a}}{w_{i,i}}, \tag{F.5}$$

where a is a nonnegative integer and division is element-wise. The source coefficients are defined by Equation F.2 for $i = 1$ and Equation F.3 for $i \geq 2$. For either case, $L_i(a) = \prod_{k=i+1}^{i+a} z_k$, with $L_i(0) = 1$. For multi-segment semantic sessions, subsequent segment retentions are composed over the corresponding event boundaries. For the scalar survival curves, we divide the coordinate mean of the remaining source coefficient by its coordinate mean at insertion. This weights coordinate-wise survival by the source's initial write coefficients. At each session age, eligible sources are those whose recorded trajectory extends to that age. Their scalar values are averaged within user and then equally across eligible users. All five checkpoints below include ten users. The emergence-source count is 810 at each checkpoint; supplement-source counts are 2,871 at ages 1, 5, and 10, 2,502 at age 40, and 1,782 at age 60. Table F.3 reports the numerical checkpoints underlying the survival curves in the main text.

Table F.3: Mean source survival after subsequent session updates.

Source role	1	5	10	40	60
Domain emergence	0.294	0.121	0.081	0.038	0.032
Supplement	0.225	0.058	0.028	0.008	0.005

At segment resolution, the untrained gate follows the constant decay $\sigma(-2)^a \approx 0.119^a$ and reaches 2.41×10^{-5} after five gated updates. A semantic-session horizon composes all intervening segment updates. At the five-session checkpoint, the learned recurrence retains 0.121 of emergence sources and 0.058 of supplement sources.

G Full Cross-Backbone Transfer Results

G.1 Transfer Protocol and Paired Evaluation

Source and target modules. Head Transfer reuses the session encoder from the Qwen3-8B compiler at update 51,885. Its context encoder E_ξ and Perceiver-based resampler G_ϕ remain fixed while a target-specific decoder $D_{\beta'}$ is trained for the frozen target backbone $f_{\theta'}$. From Scratch initializes the resampler and target decoder anew and optimizes both, with the context encoder and target backbone held fixed. The two conditions therefore compare reuse of the learned memory encoding with target-side learning from random initialization. At backbone replacement, the adapted decoder reads the retained memory h_T and produces $D_{\beta'}(h_T)$ for the new backbone, following Algorithm B.3. The memory shape, session encoder, and learned consolidation gate remain unchanged; the target decoder accommodates the new backbone's layer count and module dimensions.

Target-side training. Both conditions use the same 664,128 compilation sessions and fixed reference responses. Reference distributions are prepared with the corresponding target backbone conditioned on each textual session; the paired runs for that target share these cached distributions. Each run completes one pass, 10,377 updates, with global batch size 64 and the same random seed, using the fixed-reference forward-KL objective. The source compiler's five-pass training precedes this target-side comparison and supplies the checkpoint shared across transfer targets. The matched budget refers to the one-pass target training stage.

The target configurations retain eight Perceiver queries, latent width 512, LoRA rank eight, nine Perceiver encoder blocks, and a four-block decoder pre-head. Dense targets adapt the MLP down-projection in each layer. For Qwen3.5-35B-A3B, the decoder generates factors for the always-active shared expert's down-projection. The documented optimizer configuration uses AdamW with learning rate 4×10^{-5}, weight decay 0.01, 500 warmup updates, gradient clipping at 1.0, and seed 42. The factor regularization coefficient is 0.01, and the LoRA scaling coefficient is 32.

Batch construction and runtime alignment. Qwen3-4B, Ministral-3-8B, and Qwen3.5-9B use eight single-GPU data-parallel replicas, each processing one session per microbatch and accumulating eight microbatches. The Qwen3.5-35B-A3B protocol distributes each frozen backbone replica over eight GPUs and runs four replicas. Each replica accumulates 16 sessions, and trainable-module gradients are averaged across replicas. Both layouts yield 64 sessions per optimizer update, with the same batch construction for Head Transfer and From Scratch. Validation is scheduled every 5,000 updates and at the final epoch boundary; the completed one-pass checkpoint supplies the downstream compiler. For Qwen3.5 targets, reference target preparation and compiler training use matching frozen Transformers and Tokenizers runtimes. Launch-time checks compare these runtimes with the target asset manifest to preserve token and response-position alignment.

Downstream evaluation. For the paired compiler comparison, each resulting compiler is frozen before fitting its consolidation gate with the same architecture and ten-fold leave-one-user-out PERMA protocol. Gate training uses nine users in each fold, and the remaining user supplies the reported evaluation examples. This downstream task-adaptation stage is separate from training the target decoder on the compilation corpus. Table G.1 and Table G.2 report the complete paired From Scratch and Head Transfer results over all seven PERMA variants. Figure 2 reports the corresponding RPMem performance for each language model.

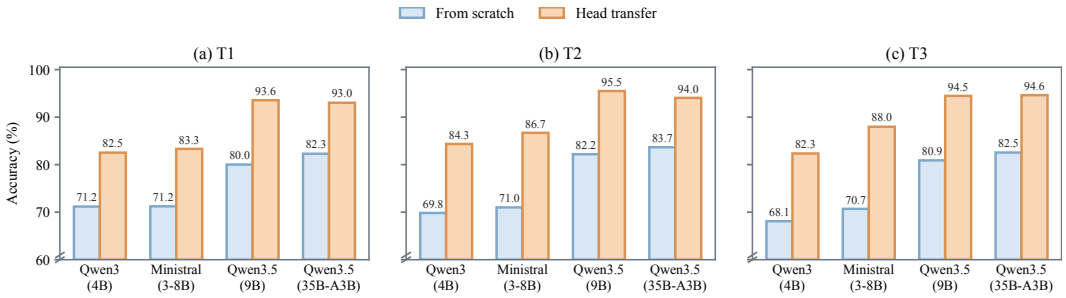


Figure G.1: Cross-backbone transfer across the PERMA memory lifecycle (accuracy, %; seven-variant average). Vertical axes begin at 60%.

Table G.1: Cross-backbone accuracy (%) on the four single-domain PERMA variants.

Target backbone	Initialization	Clean SD	Noisy SD	Style SD	Style-Long SD	SD Avg.
Qwen3-4B	From scratch	75.67	75.27	75.30	74.57	75.20
	Head transfer	**86.20**	**85.83**	**85.83**	**86.07**	**85.98**
Ministral-3-8B	From scratch	76.23	76.40	75.97	76.37	76.24
	Head transfer	**90.03**	**89.37**	**89.27**	**89.23**	**89.48**
Qwen3.5-9B	From scratch	79.10	79.57	78.57	78.53	78.94
	Head transfer	**96.50**	**96.50**	**96.00**	**95.83**	**96.21**
Qwen3.5-35B-A3B	From scratch	80.53	82.57	81.13	81.07	81.33
	Head transfer	**93.53**	**93.00**	**94.37**	**93.70**	**93.65**
Cross-model average	From scratch	77.88	78.45	77.74	77.63	77.93
	Head transfer	**91.57**	**91.18**	**91.37**	**91.21**	**91.33**

Table G.2: Cross-backbone accuracy (%) on the three multi-domain PERMA variants and all seven variants.

Target backbone	Initialization	Clean MD	Noisy MD	Style MD	MD Avg.	Overall Avg.
Qwen3-4B	From scratch	62.47	60.03	60.95	61.15	69.18
	Head transfer	**78.89**	**79.32**	**78.92**	**79.04**	**83.01**
Ministral-3-8B	From scratch	63.64	63.79	63.79	63.74	70.88
	Head transfer	**81.70**	**84.72**	**82.59**	**83.00**	**86.70**
Qwen3.5-9B	From scratch	83.33	84.63	83.97	83.98	81.10
	Head transfer	**92.52**	**92.68**	**92.57**	**92.59**	**94.66**
Qwen3.5-35B-A3B	From scratch	84.40	86.00	83.22	84.54	82.70
	Head transfer	**95.17**	**94.97**	**94.73**	**94.96**	**94.21**
Cross-model average	From scratch	73.46	73.61	72.98	73.35	75.97
	Head transfer	**87.07**	**87.92**	**87.20**	**87.40**	**89.64**

G.2 Target-Backbone Adaptation Cost

Table G.3 reports the target-side compilation stage defined above. Trainable parameters count the optimized decoder weights in Head Transfer and the resampler plus decoder weights in From Scratch. The decoder is a shared adapter-generating network; the per-user memory and its generated LoRA factors are measured separately in Appendix E.5. Time is the recorded wall-clock duration of the one-pass target training run. The first three targets use eight GPUs on one node, while Qwen3.5-35B-A3B uses 32 GPUs across four nodes. Timing comparisons are paired within each target and hardware allocation. Validation loss is the final checkpoint's fixed-reference forward-KL validation metric on the compilation data.

Table G.3: One-pass target-backbone compilation cost.

Target backbone	Initialization	Trainable params	GPUs	Time (h)	Val. loss
Qwen3-4B	From scratch	580.67M	8	7.64	0.7568
	Head transfer	**532.40M**	8	**6.74**	**0.6759**
Ministral-3-8B	From scratch	659.72M	8	8.16	0.6827
	Head transfer	**611.45M**	8	**7.46**	**0.5944**
Qwen3.5-9B	From scratch	589.67M	8	32.27	0.6486
	Head transfer	**541.40M**	8	**30.92**	**0.5815**
Qwen3.5-35B-A3B	From scratch	437.48M	32	41.66	0.5713
	Head transfer	**389.21M**	32	**38.42**	**0.5121**

Freezing the transferred resampler reduces target-side trainable parameters by 7.32% to 11.03% across the four backbones. Adaptation time decreases by 4.17% to 11.74%, and the final Fixed FKL validation loss is lower for every target. Together with the paired downstream results, these measurements show that the retained encoder improves target-side learning under the same target-data exposure.

The cost table accounts for target-side training after the source compiler and reference targets have been prepared. Source compilation is the shared upstream training investment described in Appendix E.4. Downstream gate fitting has its own per-fold cost, reported in Appendix E.5, which also defines the forward-only write and query measurements. Thus the reported training durations and deployment latencies refer to distinct stages of the memory lifecycle.

H Additional Comparisons with Metis

H.1 Comparison on Qwen3.5-9B

To control backbone family and scale in the comparison with Metis, we evaluate No Context, Full Context, Metis-9B, and RPMem on Qwen3.5-9B. All four methods share the PERMA task splits, visible-history boundary, non-thinking mode, and A–H next-token-logit answer selection. RPMem uses the adapted decoder described in Appendix G and fits its consolidation gate on the nine training users of each fold. Metis-9B uses its released memory-model weights without additional PERMA training.

Table H.1: Same-backbone comparison on PERMA with Qwen3.5-9B (accuracy, %).

Method	Clean SD	Noisy SD	Clean MD	Noisy MD	Core Avg.	7-Variant Avg.
No Context	29.20	29.20	35.63	35.63	32.41	31.98
Full Context	69.27	72.00	85.71	85.68	78.17	74.25
Metis-9B	80.87	80.47	80.17	79.29	80.20	80.04
RPMem (ours)	**96.50**	**96.50**	**92.52**	**92.68**	**94.55**	**94.66**

Core Avg. averages the four displayed settings; 7-Variant Avg. additionally includes Style SD, Style-Long SD, and Style MD. Under this shared backbone and evaluation protocol, RPMem exceeds Metis-9B by 14.35 pp on the core average and 14.62 pp across all seven variants.

H.2 Metis Scale Results

We evaluate all three released Metis scales under the same seven-variant PERMA protocol. Table H.2 shows that Metis-9B obtains the highest overall average and is therefore used in the main comparison; Metis-27B is strongest only on Clean MD.

Table H.2: PERMA accuracy (%) across released Metis model scales.

Model	Clean SD	Noisy SD	Style SD	Style-Long SD	Clean MD	Noisy MD	Style MD	Core Avg.
Metis-4B	73.77	74.67	73.27	73.13	70.81	70.71	68.34	72.49
Metis-9B	**80.87**	**80.47**	**79.67**	**79.97**	80.17	**79.29**	**79.87**	**80.20**
Metis-27B	78.67	74.87	77.03	77.03	**81.55**	78.45	78.65	78.39

I Evaluation Protocol Checks and Supplementary Diagnostics

We check prompt reproduction and answer scoring to contextualize the PERMA comparisons. Under the original prompt, Qwen3-32B reproduces the official standalone Type-3 results within 0.7 pp in each setting; Table I.1 also reports the results under our prompt. For Qwen3-8B in the No Context condition, API generation and local A–H logits differ by 0.22 pp on SD and 1.90 pp on MD.

Table I.1: Reproduction of PERMA standalone Type-3 accuracy (%) with Qwen3-32B.

Setting	Reported	Official prompt	Our prompt
Clean SD	87.00	87.00	87.00
Noisy SD	87.70	87.23	87.71
Clean MD	93.60	93.63	87.90
Noisy MD	93.00	92.36	89.17
Macro average	90.33	90.05	87.94

Query-only evaluation provides a reference for performance available without historical information. Qwen3-32B reaches 75.58–75.89% on Type 3 under this condition, and Qwen3-8B reaches 52.70–66.95% depending on domain. Table I.2 supplements these checks with our GPT-4o-mini No Context reproduction and memory-system results reported by PERMA (Liu et al., 2026b); the table distinguishes their provenance. Together, these results document protocol sensitivity and query-only performance. Attribution of the observed differences to information loss, contextual interference, or answer-option cues requires additional controlled comparisons.

Table I.2: Supplementary GPT-4o-mini results on PERMA (accuracy, %). No Context is our reproduction; all other rows are reported results from the benchmark paper.

Method	Clean SD	Noisy SD	Clean MD	Noisy MD	Avg.
No Context	73.19	73.19	**82.26**	**82.26**	77.73
Full history	78.00	76.60	70.70	72.00	74.33
RAG (BGE-M3)	70.20	71.90	68.20	66.90	69.30
MemOS	**81.10**	**85.30**	73.20	75.20	**78.70**
Mem0	68.60	66.00	65.00	66.20	66.45
LightMem	65.70	67.10	60.50	63.10	64.10
Memobase	73.30	68.30	69.40	64.30	68.83
EverMemOS	72.80	69.50	71.30	73.20	71.70
Supermemory	65.50	67.40	65.60	63.70	65.55

J Additional Gate Dynamics Visualization and Case Studies

Figure J.1 presents a deterministically selected representative trajectory: among 423 candidates with at least 40 sessions, three domains, and both emergence and supplement events, it is nearest to the multivariate median of four gate statistics. The trajectory contains 82 sessions for held-out user 507, including ten preference events from seven domains and 72 distractor sessions. The write curve displays sessions 2–82, with the first session omitted. This instance illustrates the aggregate pattern in subsection 4.2: early domain-establishing events can form a long-lived backbone even when later same-domain updates temporarily dominate.

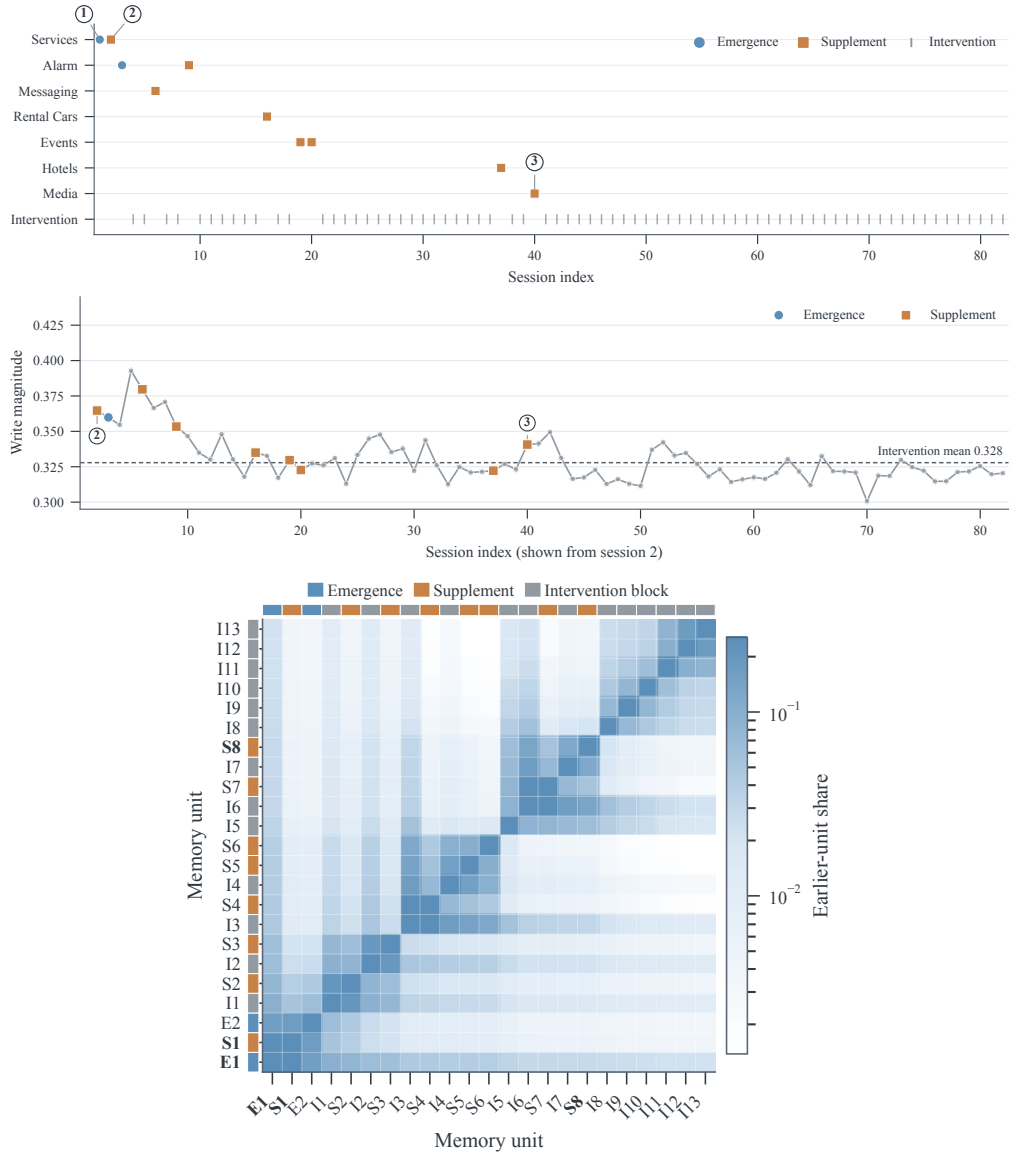


Figure J.1: Representative held-out trajectory: event timeline (top), session-wise write magnitude from session 2 onward (middle), and memory-unit source shares (bottom). The timeline and heatmap cover the complete history. E and S denote emergence and supplement events on the task's preference timeline; I denotes a block of consecutive intervention sessions. An intervention may replay a preference event from another timeline, including a domain's emergence. For each pair of units, the heatmap gives the earlier unit's summed source share at the end of the later unit, mirrored across the diagonal. Color intensity uses a logarithmic scale. Circled numbers identify three selected events.