LIFEMEMBENCH: BENCHMARKING LONG-TERM MEMORY FOR EVOLVING USER STATES THROUGH PERSONALIZED PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Personalized language-model agents increasingly interact with users over extended periods, yet user preferences, constraints, goals, and circumstances evolve over time. Agents must therefore recover relevant information, resolve what remains valid, and use the resulting user state in subsequent decisions. Existing evaluations have studied long-term retrieval, temporal reasoning, memory updates, and preference following, but often separately from the downstream decisions they are intended to support. We introduce LifeMemBench, a benchmark for *longitudinal personalization under evolving user states*. LifeMemBench constructs controlled latent user-state trajectories before realizing them as chronological interactions, providing query-time supervision for both current and historical states. It jointly evaluates state recovery, personalized planning, and replanning through a common process of *recovering* relevant information, *resolving* the valid state, and *acting* on it. Spanning 8K–64K tokens, planning and replanning are evaluated against explicit user-specific and feasibility constraints to test whether longitudinal information affects downstream behavior. We further introduce CoMem, a lightweight state-aware memory system that preserves historical evidence, tracks state changes, and derives an actionable current-state view. Under a shared Qwen3.5-4B backbone, CoMem achieves 47.23% QA correctness and 46.53%/41.24% on planning/replanning, compared with the strongest competing results of 40.27% and 33.05%/29.55%. Across systems and controlled ablations, we find that stronger information recovery does not necessarily yield better personalization: successful long-term interaction requires not only remembering user information, but also determining which state remains valid and consistently translating it into action.

1 INTRODUCTION

Large language model (LLM) assistants are evolving into persistent agents that engage with the same user over extended periods. Effective personalization requires tracking user preferences, constraints, goals, and circumstances as they accumulate and change, an ability underpinned by long-term memory that extracts, organizes, updates, and retrieves information from long interaction histories (Zhong et al., 2023; Chhikara et al., 2025), with recent frameworks introducing tailored memory designs (Xu et al., 2025; Wang & Xi, 2025; Shen et al., 2026b).

However, memorizing user information does not guarantee effective personalization: a user may revise dining tastes, travel with elderly companions, set hotel budgets, and favor public transportation, yet an agent that retrieves these details may still rely on outdated preferences or violate implicit constraints. Longitudinal personalization therefore demands three capabilities: recovering relevant historical information, resolving the currently valid user state, and acting on it.

Existing benchmarks have advanced long-context retrieval, multi-session memory, temporal reasoning, knowledge updates, and personalized preference following (Hsieh et al., 2024; Salemi et al., 2023; Maharana et al., 2024; Wu et al., 2025; Zhao et al., 2025). Recent work has further expanded memory evaluation toward dynamic user profiles, evolving or obsolete information, implicit user constraints, and fine-grained memory operations (Jiang et al., 2025a; Chen et al., 2025; Wang et al., 2026; Li et al., 2026; Shen et al., 2026a; Uddin et al., 2026; Xie et al., 2026; Fan et al., 2026). However,

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.

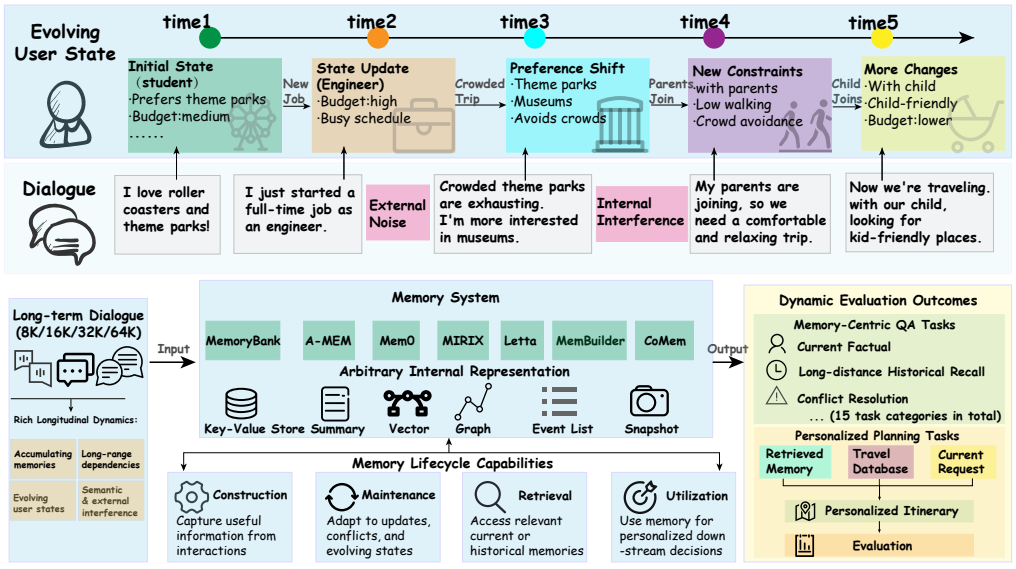

Figure 1: **Overview of LifeMemBench.** LifeMemBench evaluates longitudinal personalization through a unified *recover–resolve–act* process. Memory QA evaluates recovery and temporal resolution of evolving user information, while personalized planning and replanning test whether the resulting state guides downstream decisions.

these capabilities are less often connected within the same controlled longitudinal trajectory with explicit state supervision and downstream constraint-based evaluation. This makes it difficult to assess whether an agent can recover evolving user information, resolve the currently valid state, and reliably translate that state into personalized decisions.

To address this gap, we introduce LifeMemBench, a longitudinal personalization benchmark built upon state-first construction: it predefines complete latent user-state trajectories before generating chronological interactions. This design enables precise query-time state supervision while only exposing models to raw interaction histories, covering diverse realistic state dynamics including information addition, revision, invalidation, and context-specific suppression.

As illustrated in Figure 1, LifeMemBench unifies memory QA, personalized planning, and replanning under consistent long-term trajectories: QA validates information recovery and state resolution, planning tests downstream decision-making under user constraints, and replanning tests adaptation to user or environmental changes, jointly instantiating the *recover–resolve–act* pipeline rather than evaluating retrieval in isolation.

We instantiate LifeMemBench on travel planning, a practical scenario requiring integration of cumulative user information and verifiable real-world constraints. In this setting, user preferences over budgets, companions, mobility, transportation, dining, and accommodation continuously evolve and interact, while generated plans must maintain environmental feasibility. Our benchmark covers interaction scales from 8K to 64K, enabling systematic comparison of diverse memory systems on identical chronological interaction streams without restricting internal memory designs.

To bridge the gap between information recovery and downstream decision making, we introduce CoMem. It is a lightweight state-aware memory system that distinguishes historical evidence from the actionable current state. CoMem preserves interaction evidence, tracks state-changing events, and derives a current-state view for downstream use without discarding the history required for temporal queries. Under a shared Qwen3.5-4B backbone, CoMem achieves 47.23% QA correctness, 46.53% planning, and 41.24% replanning, compared with the strongest competing results of 40.27%, 33.05%, and 29.55%, respectively. Further analysis suggests that stronger information recovery does not necessarily yield better downstream personalization, highlighting the importance of evaluating not only what agents remember, but also how they resolve and use remembered information.

Our contributions are:

- **LifeMemBench**, a state-first benchmark for longitudinal personalization that models evolving user states under updates, conflicts, attribution, and interference across 8K–64K interactions, with query-time supervision for both current and historical states.
- A **recover–resolve–act evaluation** connecting memory QA, personalized planning, and replanning within the same trajectories, testing whether recovered user information is correctly resolved and translated into downstream decisions under explicit personalization and feasibility constraints.
- **CoMem**, a lightweight state-aware memory system that preserves historical evidence while maintaining an actionable current-state view, together with empirical analyses exposing a gap between information recovery and downstream personalization.

2 Related Work

Agent memory systems. Long-term language-model agents use external memory to preserve information across extended interactions. MemoryBank and Mem0 compress conversation histories into summaries or salient facts, whereas A-MEM, MIRIX, and Letta organize memories through linked, specialized, or hierarchical structures (Zhong et al., 2023; Chhikara et al., 2025; Xu et al., 2025; Wang & Xi, 2025; Packer et al., 2023; Letta, 2026). MemBuilder further learns which information to retain through reinforcement learning (Shen et al., 2026b). These systems use different memory representations. Less structured memories mainly store summaries or retrieved facts, while structured memories explicitly represent events, relations, and state transitions, making temporal updates and conflict resolution easier to handle.

Personalization and evolving user states. Personalization requires tracking preferences and constraints as they change across interactions. PERMA models preference development through ordered events (Liu et al., 2026), while PersonaMem evaluates dynamic user profiling and responses to the user's current state (Jiang et al., 2025a). PersonaMem-v2 extends this direction to implicitly revealed preferences and agentic memory in long-context interactions (Jiang et al., 2025b). RPEval and Know It, Act on It further show that recalling a preference does not guarantee that it will be used appropriately (Feng et al., 2026a;b). LifeMemBench studies this recover–resolve–act process under longitudinal updates, conflicts, and changing user constraints.

Planning benchmarks. TravelPlanner, ChinaTravel, and Flex-TravelPlanner evaluate itinerary construction under explicit or externally provided constraints (Xie et al., 2024; Shao et al., 2026; Oh et al., 2025). Trip+ further considers stateful travel planning (Chen et al., 2026). These benchmarks usually provide task constraints through the current request, dialogue, or environment. LifeMemBench instead requires agents to recover constraints from preceding longitudinal interactions, determine which revisions and conflicts are still valid, and use the resulting state for personalized planning and replanning.

3 LifeMemBench: Longitudinal Personalization under Evolving User States

3.1 Benchmark Overview

LifeMemBench evaluates whether agents remain personalized as users evolve over long-term interactions. Building on longitudinal memory evaluation (Maharana et al., 2024; Wu et al., 2025), it goes beyond retrieval to test whether agents can recover relevant information, resolve the valid user state, and use it in downstream decisions. We operationalize this *recover–resolve–act* process through memory QA, personalized planning, and replanning over shared longitudinal trajectories.

For each user, LifeMemBench provides a chronological interaction stream $\mathcal{D} = \{d_1, \ldots, d_T\}$. A task at position q observes only $\mathcal{D}_{<q} = \{d_1, \ldots, d_{q-1}\}$, while while historical queries additionally specify a target time. LifeMemBench is *representation-agnostic*: systems may use summaries, facts, vectors, graphs, profiles, or other memory structures while receiving matched observations and task positions. The online protocol and system-visible inputs are detailed in Appendix A.7 and Appendix P.3.

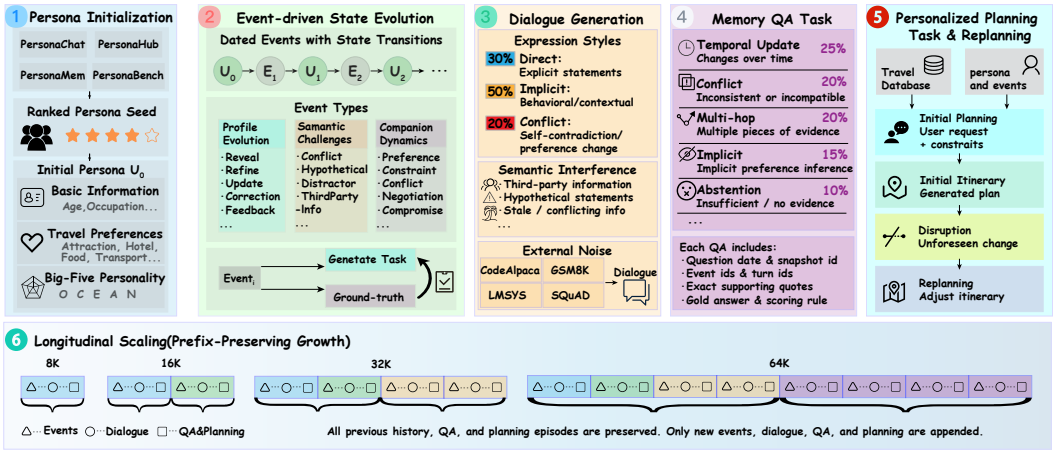

Figure 2: **State-first construction of LifeMemBench.** Latent user states evolve through temporally ordered events and are realized as chronological dialogue with controlled interference. Memory QA, planning, and replanning are derived from the same trajectories. Longer scales preserve earlier history while appending new interactions and state changes.

3.2 State-First Longitudinal Construction

Longitudinal personalization requires distinguishing currently valid information from information that is merely relevant or historically true (Wu et al., 2025; Uddin et al., 2026; Xie et al., 2026; Fan et al., 2026). LifeMemBench therefore constructs the evolving user state *before* realizing it as dialogue.

Each trajectory begins with a latent user profile and temporally ordered events. An event e_t updates the valid state as $S_t = F(S_{t-1}, e_t)$. Events are then realized as natural multi-turn interactions. Latent profiles, event annotations, and state snapshots are used only to construct and evaluate the benchmark; systems observe only chronological dialogue. Details are provided in Appendix A.4.

Interactions may contain revisions, conflicts, superseded preferences, historical experiences, hypothetical statements, companion information, and distractors, not all of which alter persistent state. The trajectory therefore records *whose* information is expressed, *when* it applies, and *whether* it changes the valid state. This separates semantic relevance from state validity.

Figure 2 summarizes the construction process. Data schema, state representation, event roles, and evidence control are detailed in Appendix A, with a complete example in Appendix D and validity checks in Appendix O. A detailed illustration of the longitudinal dialogue stream, state updates, historical queries, and replanning tasks is provided in Figure 4. Longer scales use *prefix-preserving growth*, adding interactions, state changes, tasks, and interference over the same users.

3.3 From State Recovery to Personalized Action

Memory QA: recovering and resolving user state. Memory QA tests whether information can be recovered and resolved under the appropriate temporal scope (Maharana et al., 2024; Wu et al., 2025). *Current-state* questions target the state valid at query time, whereas *historical-state* questions reconstruct an earlier state after subsequent updates.

We evaluate six capabilities: simple retrieval, implicit memory, multi-hop reasoning, temporal update, conflict resolution, and abstention. The first three primarily test recovery and composition; temporal-update and conflict tasks require state resolution, while abstention tests unsupported state construction. The 15 fine-grained task types are mapped in Appendix C.

Planning: acting on the resolved state. LifeMemBench uses personalized travel planning to test whether recovered user information affects action. Following constraint-grounded evaluation (Xie et al., 2024; Shao et al., 2026), plans must satisfy accumulated budget, mobility, transportation,

Table 1: Scale of LifeMemBench. Longer tiers extend the same 16 user trajectories.

Scale	Users	Events	Turns	QA	Planning
8K	16	128	2,084	160	32
16K	16	256	4,096	320	64
32K	16	512	8,032	640	128
64K	16	1,024	15,266	1,280	256

accommodation, dining, and companion requirements while remaining feasible in a shared structured environment.

Planning combines the current request, the structured travel environment, and the user information recovered from memory. Figure 5 provides an illustrative example of how these sources jointly influence personalized planning choices and replanning.

Planning is evaluated with atomic constraints rather than similarity to a single reference itinerary. Criterion provenance distinguishes requirements from longitudinal history, the current request, and the environment, separating history-dependent personalization from generic instruction following. Constraints are checked deterministically against structured attributes when possible; semantic criteria use criterion-level rubrics with a structured LLM judge. Full planning and replanning construction details are provided in Appendix E.

Replanning: adapting as the state changes. Replanning introduces a new circumstance after an initial plan, such as resource unavailability, transport disruption, a budget revision, or a new companion constraint. The agent must revise invalid decisions while preserving valid requirements. This tests adaptation to subsequent changes. Details are provided in Appendix E.

3.4 Longitudinal Scales and Evaluation Protocol

LifeMemBench contains four cumulative scales—8K, 16K, 32K, and 64K—over the same 16 users. Prefix-preserving growth progressively adds state changes, superseded or competing information, and distractors. Complete statistics are provided in Appendix B.

Table 1 summarizes the benchmark scale at each interaction length.

Excluding age-only changes, the four tiers contain 79, 146, 282, and 568 state-changing events. Scaling therefore measures robustness under increasing *longitudinal pressure*, rather than context length alone.

All systems receive the same chronological streams and task positions. Each system retains its native memory representation and update procedure. Privileged annotations are used only for evaluation and never exposed to tested systems.

Scope and generalizability. Although instantiated in travel planning, LifeMemBench's construction is not tied to travel-specific semantics. Its state-transition and atomic-constraint interfaces can be adapted by replacing domain-specific states, constraints, and environments while retaining the evaluation protocol. Empirical validation beyond travel remains future work.

4 CoMem: A Lightweight State-Aware Memory System

We introduce CoMem, a lightweight state-aware memory system for longitudinal personalization. Its design addresses a central tension: raw dialogue preserves historical evidence but obscures the valid current state, while an overwritten profile provides a compact current view but may discard history needed for temporal queries (Packer et al., 2023; Xu et al., 2025; Wang & Xi, 2025; Shen et al., 2026b).

CoMem therefore maintains three complementary views: interaction memory $\mathcal{C}$ preserves the original conversational evidence; event memory $\mathcal{E}$ records attributed state changes without destructive overwriting; and a derived state $\mathcal{S}$ resolves the currently valid user information for downstream action.

State updates modify $\mathcal{S}$ while their historical evidence in $\mathcal{C}$ and $\mathcal{E}$ remains available. The end-to-end algorithm and implementation details are provided in Appendix F.

4.1 Memory Construction

At each step, CoMem processes only the interaction prefix available before the task, first passing incoming turns through a conservative deterministic *noise gate* that removes only high-confidence irrelevant content and retains uncertain information. A Qwen3.5-4B extraction call with deterministic decoding maps each retained turn into

$$e_i = (m_i, s_i, r_i, o_i, \tau_i, i), \tag{1}$$

where m_i is the preserved memory content, s_i its attributed subject, r_i the source role, $o_i \in \{\texttt{ADD}, \texttt{UPDATE}, \texttt{REMOVE}, \texttt{NONE}\}$ the update operation, τ_i an optional timestamp, and i the stream position. Events are appended to $\mathcal{E}$ without destructive overwriting. Filtering rules, and the complete extraction prompt, output schema, validation procedure, and failure handling, are provided in Appendices F.3 and F.4, respectively.

Role attribution distinguishes user, companion, third-party, and assistant information, and hypothetical statements are preserved rather than promoted to user state. Adjacent dialogue turns are also retained in $\mathcal{C}$ as original evidence to limit information loss during extraction (Appendix F.1).

4.2 State Maintenance

CoMem resolves evolving information while enforcing its temporal scope (Wu et al., 2025).

Temporal guard. Before state resolution or retrieval, a *temporal guard* prevents future leakage. For a task at position q with optional target time t_q, eligible events are

$$\mathcal{E}^{(q)} = \{e_i \in \mathcal{E} \mid i < q \wedge (t_q = \varnothing \vee \tau_i = \varnothing \vee \tau_i \leq t_q)\}. \tag{2}$$

The same constraint applies to $\mathcal{C}$. Filtering is performed before lexical or semantic scoring, so future evidence cannot enter candidate generation or ranking. For undated memories, stream position enforces causality. See Appendix F.5 for details.

Derived current state. From eligible events, CoMem deterministically derives an actionable state

$$\mathcal{S}_q = G(\mathcal{E}^{(q)}), \tag{3}$$

without an additional model call. G retains non-hypothetical, state-relevant user events and organizes them into coarse slots such as budget, transportation, accommodation, food, attractions, and accessibility. Within each slot, the latest eligible state-changing event determines the current value, and the resulting state is filtered for request relevance.

Only the derived state is updated. Superseded events remain in $\mathcal{E}$ for historical queries. CoMem thus maintains an actionable current state without discarding state-transition evidence (Xu et al., 2025; Shen et al., 2026b). State eligibility and snapshot details are provided in Appendix F.2.

4.3 Time-Safe Retrieval

Retrieval operates only over temporally eligible evidence. Structured events are ranked by semantic similarity, lexical relevance, recency, and entity overlap using $\mathrm{score}(e, q) = \lambda_s s_{\mathrm{sem}} + \lambda_l s_{\mathrm{lex}} + \lambda_r s_{\mathrm{rec}} + \lambda_e s_{\mathrm{entity}}$. For historical queries, the recency term is disabled to avoid favoring the current state over temporally relevant evidence.

For QA, interaction blocks provide primary evidence, with structured events as fallback. For planning, event memories provide personalized constraints together with the derived current state. Retrieval details are reported in Appendix F.6.

Overall, CoMem separates evidence preservation, state-change tracking, and current-state resolution while remaining lightweight: only event extraction requires a model call; noise filtering, temporal guarding, interaction construction, and state derivation are deterministic.

Table 2: **Overall performance on LifeMemBench.** All systems use Qwen3.5-4B. Higher is better.

Method	QA RC ↑	Planning ↑	Replanning ↑
Mem0	0.319	0.307	0.296
A-MEM	0.359	0.301	0.277
Letta	0.372	0.295	0.244
MemBuilder	0.356	0.318	0.271
MemoryBank	0.403	0.331	0.264
MIRIX	0.365	0.322	0.286
CoMem	**0.472**	**0.465**	**0.412**

5 EXPERIMENTS

We evaluate LifeMemBench under a shared Qwen3.5-4B generation backbone. Each method retains its native memory representation and update mechanism, while inputs, task positions, downstream interfaces, and evaluation protocols are standardized. We study four questions:

RQ1: Can memory systems recover, resolve, and act on evolving user states?

RQ2: How robust are memory systems to state changes and growing histories?

RQ3: Does longitudinal memory improve personalized planning beyond the current request?

RQ4: Which CoMem components drive state resolution and downstream action?

5.1 EXPERIMENTAL SETUP

Shared backbone and interface. We compare representative long-term memory systems covering fact-based designs (Zhong et al., 2023), structured memory architectures (Chhikara et al., 2025; Packer et al., 2023; Letta, 2026), and dynamic evolving memory paradigms (Xu et al., 2025; Wang & Xi, 2025; Shen et al., 2026b). All systems process the same chronological streams with Qwen3.5-4B (Qwen Team, 2026) as the shared generator while retaining their native memory mechanisms. Retrieved memory is exposed through a standardized interface with at most five personal-memory items ($K = 5$); the shared travel database returns at most 35 records. Planning receives the request, memory, and database; replanning also receives the previous plan and observed change. We do not add a CoMem-style state snapshot to competing systems. Baseline settings are provided in Appendix G; metric and judge details are in Appendices H and I.The released benchmark repository provides the database schema and files, deterministic evaluation code, and the sampling, filtering, injection, and truncation pipeline at `https://anonymous.4open.science/r/LifeMemBench-5B33/`.

Statistical protocol. We estimate uncertainty at the user level using clustered bootstrap confidence intervals over the 16 users and paired tests for direct comparisons. Full results are in Appendix N, with reproducibility details in Appendix P.

5.2 OVERALL PERFORMANCE

To RQ1: From recovery to personalized action. Table 2 reports QA Response Correctness (RC), planning, and replanning; complete results are provided in Appendix J, with statistical tests in Appendix N.

CoMem performs best across all three tasks, with particularly large gains on planning and replanning. Baseline rankings vary across tasks: MemoryBank is the strongest non-CoMem method for QA and planning, whereas Mem0 performs best on replanning, indicating that answer-level recovery and downstream personalization need not improve together.

Table 3: **Performance on dynamic user-state challenges.** UMR denotes Unsupported Memory Rate (↓).

Method	Multi. ↑	Conflict ↑	Temporal ↑	UMR ↓
Mem0	0.205	0.277	0.130	0.676
A-MEM	0.205	0.347	0.194	0.610
Letta	0.216	0.444	0.195	0.596
MemBuilder	0.233	0.320	0.182	0.617
MemoryBank	0.212	0.446	0.254	0.546
MIRIX	0.298	0.317	0.183	0.619
CoMem	**0.440**	**0.531**	**0.348**	**0.433**

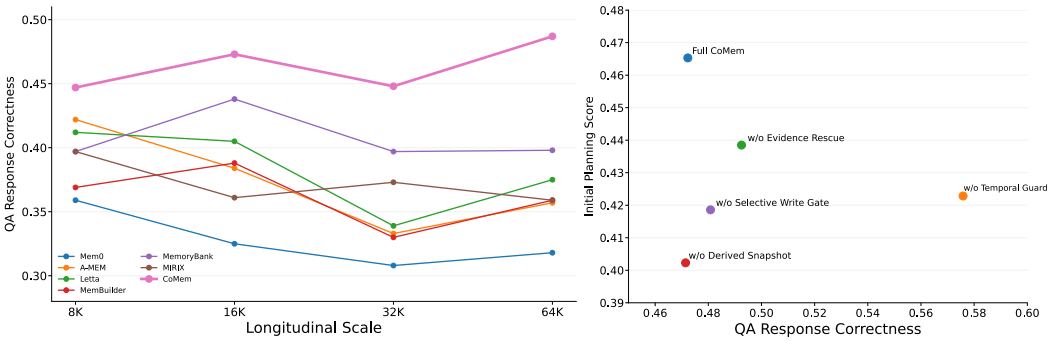

Figure 3: **Longitudinal robustness and component-level behavior. Left:** QA RC across cumulative interaction scales. **Right:** QA RC versus planning performance for CoMem and its ablations, showing that stronger recovery does not necessarily imply stronger downstream planning.

5.3 Resolving Evolving User States

To RQ2: Dynamic state resolution and longitudinal robustness. Following prior evaluations of knowledge updates and evolving user information (Wu et al., 2025; Uddin et al., 2026; Xie et al., 2026), we evaluate multi-memory integration, conflict and temporal resolution, unsupported memory generation, and robustness across cumulative scales.

As shown in Table 3, CoMem performs best across all dynamic subsets, reaching $0.440/0.531/0.348$ on multi-memory, conflict, and temporal questions and reducing UMR to 0.433. It also remains stable across increasing interaction scales (Figure 3, left). Full results are in Appendix J.2.

5.4 Does Longitudinal Information Improve Personalized Decisions?

To RQ3: Value of longitudinal user information. Following constraint-grounded travel-planning evaluation (Xie et al., 2024; Shao et al., 2026), we test whether longitudinal information improves planning beyond prompt-visible requirements. CoMem achieves 0.805 HCR, 0.743 PC, and 0.723 Feasibility, versus the strongest non-CoMem results of 0.701, 0.642, and 0.587. Full results are in Appendix J.4.

No-memory control. We evaluate Qwen3.5-4B with the same request, database, generator interface, and evaluator but without preceding interactions.

As shown in Table 4, the no-memory control reaches $0.254/0.250$ on planning/replanning, versus $0.465/0.412$ for CoMem ($+21.1/+16.2$ points), demonstrating substantial value from longitudinal information. Full results are in Appendix J.5.

Robustness across users. CoMem outperforms MemoryBank, the strongest non-CoMem planning baseline, on 15 of 16 users, with a paired improvement of 13.48 points ($p < 0.001$); user-level tests are in Appendix N.

Table 4: **Effect of longitudinal information at 64K.** "Best baseline" denotes the strongest non-CoMem result per column.

Method	Planning ↑	Replanning ↑
Qwen3.5-4B w/o Memory	0.254	0.250
Best memory baseline	0.331	0.296
CoMem	**0.465**	**0.412**

5.5 CoMem Ablation

To RQ4: Component contributions. We ablate *Selective Write*, *Derived Snapshot*, *Evidence Rescue*, and *Temporal Gate*. Figure 3 (right) summarizes the QA–planning trade-offs; full results are in Appendix K.

Removing the Derived Snapshot leaves QA RC nearly unchanged ($47.23 \rightarrow 47.15$) but reduces planning from 46.53 to 40.23, indicating that an actionable state is particularly important for downstream decisions. This further distinguishes information recovery from maintaining state suitable for action; additional component analysis is provided in Appendix K.2.

From recovery to action. Across methods, users, and interaction scales, QA correctness is only weakly associated with planning performance ($r = 0.098$, $\rho = 0.071$; Table 34 in Appendix L.1), suggesting that successful recovery does not necessarily ensure effective downstream use. Further memory–action and failure analyses, including the full QA–planning association and the four-stage memory failure taxonomy, are provided in Appendices L and M.

6 Conclusion

We introduced LifeMemBench, a state-first benchmark for longitudinal personalization under evolving user states. By jointly evaluating memory QA, planning, and replanning over controlled 8K–64K trajectories, LifeMemBench connects user-state recovery and resolution with downstream action.

We further introduced CoMem, a lightweight state-aware memory system that preserves historical evidence while maintaining an actionable current-state view. Under a shared Qwen3.5-4B backbone, CoMem improves over the evaluated memory-system baselines across QA, planning, and replanning.

Our results highlight a broader distinction between *remembering* and *using* user information: stronger recovery does not necessarily yield better downstream personalization. Evaluating persistent agents therefore requires testing not only what they remember, but whether they resolve the currently valid user state and make it consequential for subsequent decisions.

AI use statement

Generative AI tools were used for synthetic data generation, code assistance, experimental analysis, and manuscript editing. All outputs were reviewed and verified by the authors.

Reproducibility statement

Benchmark construction, experimental settings, and evaluation details are provided in the main text and Appendix. Code, data, and evaluation scripts will be released to support reproducibility.

References

Ding Chen, Simin Niu, Kehang Li, Peng Liu, Xiangping Zheng, Bo Tang, Xinchi Li, Feiyu Xiong, and Zhiyu Li. HaluMem: Evaluating hallucinations in memory systems of agents. *arXiv preprint arXiv:2511.03506*, 2025. URL https://arxiv.org/abs/2511.03506.

Junle Chen, Wei Chen, Yehong Xu, Zhengjun Huang, Yuqian Wu, Zhoujin Tian, Kai Wang, Lei Wang, and Xiaofang Zhou. Trip+: Benchmarking agents in personalized interactive travel planning. *arXiv preprint arXiv:2606.21169*, 2026. URL https://arxiv.org/abs/2606.21169.

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

Xinyi Fan, Miri Liu, Ruozhen Yang, Siru Ouyang, and Jiawei Han. Can agent memory systems track evolving state? *arXiv preprint arXiv:2608.19652*, 2026. URL https://arxiv.org/abs/2608.19652.

Xueyang Feng, Weinan Gan, Xu Chen, Quanyu Dai, and Yong Liu. How does personalized memory shape LLM behavior? benchmarking rational preference utilization in personalized assistants. *arXiv preprint arXiv:2601.16621*, 2026a. URL https://arxiv.org/abs/2601.16621.

Zhaoxin Feng, Jianfei Ma, and Emmanuele Chersoni. Know it, act on it: Investigating memory utilization in LLM personalization. *arXiv preprint arXiv:2607.29433*, 2026b. URL https://arxiv.org/abs/2607.29433.

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

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

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. *arXiv preprint arXiv:2512.06688*, 2025b. URL https://arxiv.org/abs/2512.06688.

Letta. Letta: Stateful agents with persistent memory. Software repository, 2026. URL https://github.com/letta-ai/letta. Accessed September 2026.

Yifei Li, Weidong Guo, Lingling Zhang, Rongman Xu, Muye Huang, Hui Liu, Lijiao Xu, Yu Xu, and Jun Liu. Locomo-Plus: Beyond-factual cognitive memory evaluation framework for LLM agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25085–25100. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1150. URL https://aclanthology.org/2026.acl-long.1150/.

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. *arXiv preprint arXiv:2603.23231*, 2026. URL https://arxiv.org/abs/2603.23231.

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

Juhyun Oh, Eunsu Kim, and Alice Oh. Flex-TravelPlanner: A benchmark for flexible planning with language agents. *arXiv preprint arXiv:2506.04649*, 2025. URL https://arxiv.org/abs/2506.04649.

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

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5. Official technical blog; model card at https://huggingface.co/Qwen/Qwen3.5-4B.

Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. LaMP: When large language models meet personalization. *arXiv preprint arXiv:2304.11406*, 2023. URL https://arxiv.org/abs/2304.11406.

Jie-Jing Shao, Bo-Wen Zhang, Xiao-Wen Yang, Baizhi Chen, Si-Yu Han, Jinghao Pang, Wen-Da Wei, Guohao Cai, Zhenhua Dong, Lan-Zhe Guo, and Yu-Feng Li. ChinaTravel: An open-ended travel planning benchmark with compositional DSL-based validation for language agents. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2412.13682.

Ye Shen, Dun Pei, Yiqiu Guo, Junying Wang, Yijin Guo, Zicheng Zhang, Qi Jia, Jun Zhou, and Guangtao Zhai. EvolMem: A cognitive-driven benchmark for multi-session dialogue memory. *arXiv preprint arXiv:2601.03543*, 2026a. URL https://arxiv.org/abs/2601.03543.

Zhiyu Shen, Ziming Wu, Fuming Lai, Shaobing Lian, and Yanghui Rao. MemBuilder: Reinforcing LLMs for long-term memory construction via attributed dense rewards. *arXiv preprint arXiv:2601.05488*, 2026b. URL https://arxiv.org/abs/2601.05488.

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

Yihao Wang, Haoran Xu, Renjie Gu, Yixuan Ye, Xinyi Chen, Xinyu Mu, Yuan Gao, Chunxiao Guo, Peng Wei, Jinjie Gu, Huan Li, Ke Chen, and Lidan Shou. MedMemoryBench: Benchmarking agent memory in personalized healthcare. *arXiv preprint arXiv:2605.11814*, 2026. URL https://arxiv.org/abs/2605.11814.

Yu Wang and Chen Xi. MIRIX: Multi-agent memory system for LLM-based agents. *arXiv preprint arXiv:2507.07957*, 2025. URL https://arxiv.org/abs/2507.07957.

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

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. TravelPlanner: A benchmark for real-world planning with language agents. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2402.01622.

Wenya Xie, Shengming Zhou, Zelin Li, Pouya Parsa, Shuang Zhou, Xinheng Ding, Chinmay Arvind, Guanchu Wang, Vladimir Braverman, Ali Payani, Yantao Zheng, and Zirui Liu. DynamicMem: A long-horizon memory benchmark in real-world settings. *arXiv preprint arXiv:2606.22877*, 2026. URL https://arxiv.org/abs/2606.22877.

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*, 2025. URL https://arxiv.org/abs/2502.12110.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 Embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025. URL https://arxiv.org/abs/2506.05176.

Siyan Zhao, Mingyi Hong, Yang Liu, Devamanyu Hazarika, and Kaixiang Lin. Do LLMs recognize your preferences? evaluating personalized preference following in LLMs. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2502.09597.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. *arXiv preprint arXiv:2305.10250*, 2023. URL https://arxiv.org/abs/2305.10250.

Table 5: High-level schema of the released LIFEMEMBENCH data.

Field	Description	Visible to Memory System?
`user_id`	Unique user/persona identifier	Yes
`dialogue`	Chronological natural-language interactions	Yes
`timestamp`	Temporal annotation associated with interactions	Yes
`stream_index`	Chronological interaction position	Yes
`event`	Structured event annotation	No
`state_before/state_after`	Latent user-state transition	No
`snapshot`	Ground-truth state after an event	No
`task`	QA/planning/replanning request	Yes, when encountered
`task_type`	Fine-grained evaluation category	No
`target_time`	Historical target time when applicable	Only when expressed by the task
`evidence`	Gold supporting interactions/events	No
`planning_constraints`	Ground-truth evaluation criteria	No
`constraint_provenance`	Information source of each planning criterion	No

A ADDITIONAL LIFEMEMBENCH CONSTRUCTION DETAILS

This appendix provides additional construction, annotation, evaluation, and implementation details for LIFEMEMBENCH. We describe the released data schema, latent state and event representation, dialogue realization, controlled interference, online task placement, planning construction, and evaluation metadata.

Unless otherwise stated, latent profiles, event annotations, state snapshots, evidence links, reference answers, and planning criteria are *privileged benchmark metadata*. Evaluated memory systems observe only the chronological interaction stream and the task information available at the corresponding evaluation point.

A.1 RELEASED DATA SCHEMA

Each trajectory is associated with a unique user identifier and contains a chronologically ordered sequence of natural-language interactions. The released benchmark additionally retains structured annotations used for generation, validation, and evaluation.

At a high level, each trajectory contains:

- **User identifier:** a unique identifier linking interactions from the same longitudinal trajectory;
- **Dialogue stream:** chronological user and assistant turns with timestamps and stream indices;
- **Latent user-state annotations:** structured user information used to control trajectory generation;
- **Event annotations:** state-revealing or state-changing events associated with dialogue;
- **State snapshots:** the valid user state after designated events;
- **Tasks:** memory QA, planning, and replanning instances inserted at selected positions in the stream;
- **Task metadata:** task family, fine-grained subtype, difficulty, question time, and target time where applicable;
- **Evidence annotations:** events and dialogue turns supporting each evaluation target; and
- **Planning annotations:** query-time constraints, reference itinerary information, constraint provenance, and structured evaluation criteria.

A critical distinction is made between *observable fields* and *privileged benchmark annotations*. Evaluated systems observe only the chronological dialogue and the task information presented at the current interaction point. Latent states, event labels, future interactions, evidence annotations, reference answers, and evaluation constraints are never exposed to the tested memory system.

Table 5 summarizes the major fields.

A.2 Latent User State and Event Representation

LifeMemBench is constructed from explicit latent user trajectories rather than by retrospectively inferring user states from generated dialogue. Each trajectory begins with a coherent seed profile containing attributes that may affect subsequent personalization, including preferences, constraints, goals, travel habits, companion information, spending tendencies, and behavioral characteristics.

The latent state at time t can be viewed as a collection of temporally valid attributes,

$$S_t = \{(k_i, v_i, \tau_i, a_i)\}_{i=1}^{N_t}, \tag{4}$$

where k_i denotes an attribute, v_i its value, τ_i its temporal scope, and a_i its attribution. Attribution distinguishes information about the primary user from information concerning companions or other third parties.

State evolution is induced by a chronological event sequence

$$\mathcal{E} = \{e_1, e_2, \ldots, e_M\}, \tag{5}$$

with

$$S_t = F(S_{t-1}, e_t). \tag{6}$$

An event does not necessarily modify the persistent user state. Its effect depends jointly on semantic content, attribution, temporal scope, and event role. This distinction prevents semantically relevant but non-actionable information from being automatically treated as part of the active user state.

For each event, the benchmark retains privileged metadata describing:

- event timestamp and chronological position;
- semantic attribute or constraint involved;
- whether the event introduces, modifies, confirms, conflicts with, or invalidates an existing value;
- whether the information concerns the user, a companion, or another entity;
- whether the information is current, historical, hypothetical, conditional, stale, or otherwise non-actionable;
- state before and after the event; and
- dialogue turns providing observable evidence for the event.

A.2.1 Event Roles and State Effects

The event space covers both ordinary accumulation of user information and cases in which semantic relevance does not imply current validity.

Table 6 summarizes the event roles used in benchmark construction.

A.3 Event-Type Distribution

Event-type distribution. Since the four interaction scales form nested prefixes of the same 16 user trajectories, the 64K tier represents the complete set of unique events, containing 1,024 events in total. We categorize events according to their observable roles in the longitudinal trajectories, based on annotated state transitions, affected subjects, and contextual information.

Among all events, 575 (56.15%) directly modify the user state, including additions or refinements, attribute updates, replacements or updates, and invalidations or removals. The remaining events provide companion-specific information, trip context, or controlled distractors. This mixture requires memory systems not only to retain newly introduced information, but also to distinguish state-changing evidence from contextual and third-party information. Table 7 reports the cumulative distribution across all four interaction scales.

Table 6: Representative event roles in LIFEMEMBENCH. Event annotations are used only for benchmark construction and evaluation.

Event Role	State Interpretation
Addition	Introduces a previously unspecified user attribute or constraint.
Update / Replacement	Changes the currently valid value of an existing attribute.
Refinement	Adds specificity without necessarily invalidating the previous value.
Confirmation	Reinforces an existing state without changing its value.
Invalidation	Explicitly removes or cancels a previously valid requirement.
Conflict	Introduces competing information whose validity must be resolved from context, attribution, or time.
Historical	Describes a previously valid state without making it current again.
Hypothetical / Conditional	Describes a possible state that should not automatically become persistent.
Companion / Third-party	Describes another person and may affect planning without becoming an attribute of the primary user.
Irrelevant	Adds conversational content without modifying the personalization state.

Table 7: Cumulative event-class distribution across interaction scales. The four scales are nested prefixes of the same 16 user trajectories; therefore, counts across columns should not be summed.

Event class	**8K**	**16K**	**32K**	**64K**
Addition / Refinement	23	60	120	221
Attribute Update	45	68	119	215
Replacement / Update	11	15	33	68
Invalidation / Removal	2	5	12	71
Third-party / Companion	29	56	113	219
Context / Trip Fact	17	50	111	224
Distractor / Noise	1	2	4	6
Total	**128**	**256**	**512**	**1,024**

A.4 STATE-TRANSITION CONSTRUCTION

For each user, we first instantiate a coherent seed profile. Subsequent events are generated conditionally on the current state and may preserve, reveal, revise, remove, or conflict with previously valid information.

The event-generation process follows three principles.

Temporal coherence. A state transition must be compatible with the information available before the event. When an attribute changes, the event should provide sufficient context for interpreting the new state.

State traceability. For state-changing events, the pipeline stores both previous and updated values. This enables current-state and historical-state tasks to be generated from the same trajectory without retrospectively reconstructing the state from dialogue.

Separation of state and observation. Not every dialogue statement modifies the user's active state. Third-party preferences, hypothetical statements, historical descriptions, assistant-provided information, and unrelated distractors may appear without producing a persistent state transition.

Table 8: External-noise injection configuration. The maximum number of injected dialogue pairs is specified per user trajectory.

Scale	Injection rate	Pairs / gap	Max. pairs / trajectory
8K	1.0	1	7
16K	1.0	1	15
32K	1.0	1	31
64K	1.0	1	63

Table 9: Realized external-noise statistics across cumulative interaction scales. Pair/turn ratio measures the fraction of final dialogue pairs originating from external sources, while token ratio measures their fraction of final dialogue tokens.

Scale	Base pairs	Noise pairs	Final pairs	Pair/turn ratio	Token ratio
8K	930	112	1,042	10.75%	14.44%
16K	1,808	240	2,048	11.72%	15.63%
32K	3,520	496	4,016	12.35%	15.84%
64K	6,625	1,008	7,633	13.21%	16.69%

A.5 External-Noise Injection

To introduce realistic off-topic interference, we augment the longitudinal dialogues with external user–assistant exchanges sampled from four English-language datasets: **CodeAlpaca**, containing programming and instruction-following interactions; **GSM8K**, containing mathematical reasoning problems; **LMSYS**, containing general user–assistant conversations; and **SQuAD**, containing reading-comprehension examples. The four sources are sampled with equal probability, followed by random example selection within the chosen source. Non-English examples are filtered out.

Injection procedure. External noise is inserted only between adjacent event-linked dialogue blocks, so that injected content does not alter the internal ordering of turns associated with an event. Each injected unit consists of one user turn and one assistant turn. Individual turns are truncated to 800 characters to prevent a single distractor from dominating the surrounding longitudinal context.

For the final benchmark, every eligible gap is considered for injection (`injection_rate`=1.0), with one external dialogue pair inserted per selected gap. Injection is capped according to interaction scale, as shown in Table 8. Because `injection_rate` denotes the probability of selecting an eligible event gap rather than the fraction of dialogue occupied by noise, it should not be interpreted as a token- or turn-level noise percentage.

Measured noise level. We additionally measure the realized external-noise contribution after injection rather than inferring it from the configuration parameter. Table 9 reports statistics aggregated over all 16 user trajectories. Token counts include dialogue text only and exclude QA and planning queries, metadata, identifiers, and chat-template overhead.

The final benchmark contains between 10.75% and 13.21% external noise when measured by dialogue pairs (equivalently, by dialogue turns), and between 14.44% and 16.69% when measured by dialogue tokens. The difference arises from variation in the lengths of the injected and original dialogue turns.

Formally, the reported pair/turn and token ratios are

$$r_{\text{pair}} = \frac{N_{\text{noise pairs}}}{N_{\text{base pairs}} + N_{\text{noise pairs}}}, \tag{7}$$

and

$$r_{\text{token}} = \frac{N_{\text{noise tokens}}}{N_{\text{final dialogue tokens}}}. \tag{8}$$

Table 10: Realized distribution of injected external dialogue pairs by source. Percentages are computed within each interaction scale.

Scale	CodeAlpaca	GSM8K	LMSYS	SQuAD
8K	31 (27.68%)	33 (29.46%)	27 (24.11%)	21 (18.75%)
16K	63 (26.25%)	45 (18.75%)	60 (25.00%)	72 (30.00%)
32K	146 (29.44%)	136 (27.42%)	93 (18.75%)	121 (24.40%)
64K	249 (24.70%)	247 (24.50%)	280 (27.78%)	232 (23.02%)

Table 11: Evidence-annotation audit across cumulative interaction scales. The first three rows form a mutually exclusive partition of the released QA tasks. "Unavailable after prefix" counts tasks for which any annotated supporting evidence occurs after the query insertion point.

Evidence category	8K	16K	32K	64K
Single-turn evidence	82	163	332	679
Multi-turn evidence	72	144	283	559
No explicit turn-level annotation	6	13	25	42
Total QA	**160**	**320**	**640**	**1,280**
Unavailable after prefix	0	0	0	0

The internal `noise20` identifier therefore denotes the target configuration used during dataset construction rather than an exact realized 20% noise fraction. We report the measured ratios above as the actual external-noise levels of the released benchmark.

Noise-source distribution. Although the four source datasets are selected uniformly in expectation, finite random sampling produces small deviations from an exact 25% split. Table 10 reports the realized source distribution. At the full 64K scale, the 1,008 injected dialogue pairs remain approximately balanced across the four sources.

The complete sampling, filtering, injection, and truncation implementation is provided with the released benchmark construction pipeline at `https://anonymous.4open.science/r/LifeMemBench-5B33/`.

A.6 Evidence-Availability Audit

A central requirement of LifeMemBench is that a memory task should be answerable from information observable at the time the query is issued. We therefore audit the released QA tasks with respect to both *evidence annotation* and *query-time availability*. For each QA instance, we inspect its linked dialogue evidence and the position at which the task is inserted into the chronological trajectory.

Evidence annotation coverage. We divide the released QA tasks into three mutually exclusive groups. *Single-turn evidence* denotes instances linked to one dialogue turn; *multi-turn evidence* denotes instances linked to multiple dialogue turns; and *no explicit turn-level annotation* denotes instances for which no dialogue turn is explicitly linked in the released annotation. The last category should not be interpreted as corrupted or necessarily unanswerable examples: some tasks, particularly abstention-oriented cases, are intentionally constructed without positive supporting evidence.

Table 11 reports the resulting audit. At the full 64K scale, 679 of 1,280 QA tasks (53.05%) have single-turn evidence and 559 (43.67%) have evidence linked across multiple turns. Together, 1,238 tasks (96.72%) therefore contain explicit turn-level evidence annotations. The remaining 42 tasks (3.28%) contain no explicit positive turn-level evidence annotation.

Query-time evidence availability. We further verify that annotated supporting evidence is observable under the online evaluation protocol. For every QA task with explicit evidence links, we compare the positions of all cited dialogue turns with the task insertion point. Across all four interaction scales, we find no released instance whose annotated supporting evidence occurs after the observable query prefix. At the full 64K scale, this corresponds to 0 of 1,280 released QA tasks.

Table 12: State-changing events after excluding age-only transitions.

Scale	8K	16K	32K	64K
State-changing events	79	146	282	568

This property is also enforced during benchmark construction: a candidate task is accepted only when its required annotated evidence is available no later than the query insertion point. Consequently, evaluated memory systems receive only the chronological prefix preceding each task and never require access to future dialogue turns to recover annotated evidence.

Evidence annotation versus answer recoverability. The audit above measures whether explicit evidence links are present and temporally available; it is distinct from requiring the gold answer to appear verbatim in a cited dialogue span. Some QA tasks require resolving information distributed across turns, temporal updates, conflicts, or implicit state changes, and abstention tasks may intentionally lack positive evidence. We therefore treat turn-level evidence availability and answer recoverability as separate properties rather than using lexical occurrence of the gold answer as an evidence-validity criterion.

Excluded candidates. Candidates whose required annotated evidence would occur outside the observable query prefix are rejected during construction and are not included in the released benchmark. The construction pipeline did not retain a complete attempt-level audit log of all rejected intermediate generations; we therefore do not retrospectively estimate the number or proportion of pre-release candidates excluded by this check. The released set itself, however, contains no detected query-prefix evidence violations. The complete validation and insertion procedures are provided in the released construction pipeline at `https://anonymous.4open.science/r/LifeMemBench-5B33/`.

A.7 Online Task Placement and Evidence Control

Evaluation tasks are inserted directly into the chronological interaction stream. For a task Q_q issued at position q, the evaluated system has access only to

$$\mathcal{D}_{<q} = \{d_1, \ldots, d_{q-1}\}. \tag{9}$$

Future dialogue turns, future state snapshots, and later events are never available to the evaluated system. Each task stores links to the event or events from which its target is derived and, where applicable, to the corresponding observable dialogue evidence.

For current-state queries, the target is determined from the state valid at the query position. Historical-state queries additionally specify a target time t_q. Their eligible event set is

$$\mathcal{E}_q^{\text{valid}} = \{e_i \mid t(e_i) \leq t_q\}. \tag{10}$$

The same attribute may therefore have several legitimate values across a complete trajectory. The correct target depends not only on the attribute but also on the temporal scope specified by the query.

B Dataset Statistics

The four interaction scales are constructed through *prefix-preserving growth* over the same 16 user trajectories. We therefore report not only nominal interaction length but also the number of dialogue turns, state-changing events, evaluation tasks, and other sources of longitudinal memory pressure at each cumulative scale.

State-changing events. After excluding events whose only state difference is age progression, the benchmark contains 79, 146, 282, and 568 genuine state-changing events at 8K, 16K, 32K, and 64K, respectively.

Table 12 reports the resulting state-changing event counts.

Table 13: Dataset statistics across cumulative interaction scales.

Statistic	**8K**	**16K**	**32K**	**64K**
Users / trajectories	16	16	16	16
Events	128	256	512	1,024
Dialogue turns	2,084	4,096	8,032	15,266
State-changing events	79	146	282	568
Total QA tasks	160	320	640	1,280
Planning tasks	32	64	128	256
Unique QA subtype strings	56	130	217	323
Average tokens	6,452	12,847	25,634	51,181
Temporal-update QA	48	80	160	320
Conflict QA	32	64	128	256
Historical QA	114	205	427	914
Replanning tasks	16	32	64	128
External-noise token ratio	14.44%	15.63%	15.84%	16.69%

Table 14: Constraint categories present in the 128 analyzed initial-planning instances. A single instance may contain multiple categories.

Category	Count	Share	Category	Count	Share
Budget / price	124	96.9%	Dietary / food preference	97	75.8%
Route / origin–destination	119	93.0%	Transport mode	94	73.4%
Trip duration	117	91.4%	Hotel / accommodation	86	67.2%
Companion / group	104	81.3%	Cultural / nature / activity	58	45.3%
Pace / crowd / relaxation	104	81.3%			

B.1 Overall Scale Statistics

Table 13 gives the complete scale-level statistics.

B.2 Planning Constraint Distribution

The analyzed initial-planning instances intentionally combine multiple requirements. Table 14 reports the frequency of major constraint categories. Categories are not mutually exclusive.

B.3 Replanning Disruption Distribution

Table 15 summarizes the replanning disruption types.

C Complete Memory QA Taxonomy

The main paper consolidates the released QA tasks into six diagnostic capabilities: simple retrieval, implicit memory, multi-hop reasoning, temporal update, conflict resolution, and abstention. The released benchmark retains the original 15 fine-grained task labels for auditability and detailed analysis.

Table 16 summarizes the six QA capabilities.

C.1 Fine-Grained QA Task Taxonomy

Although the benchmark construction pipeline supports diverse question templates and surface forms, the released QA tasks are organized into six capability-oriented categories. These categories capture the main challenges of longitudinal memory: retrieving previously observed information, inferring implicit user states, integrating distributed evidence, tracking temporal updates, resolving conflicts, and recognizing when the observable history does not support an answer. Table 17 summarizes the final taxonomy.

Table 15: Replanning disruption groups over 128 analyzed replanning instances.

Disruption Group	Count	Share
Resource availability	36	28.1%
Transportation / itinerary	32	25.0%
Companion / preference	23	18.0%
Health / mobility	22	17.2%
Budget	11	8.6%
Trip duration	4	3.1%
Total	128	100.0%

Table 16: Diagnostic QA capabilities in LIFEMEMBENCH.

Capability	Core Requirement	Representative Failure Mode
Simple Retrieval	Recover explicitly stated information from earlier interactions.	Relevant information is absent from retrieved memory or replaced by an unrelated statement.
Implicit Memory	Recover information expressed indirectly or through contextual evidence.	Surface text is retained but the implied user state is not recovered.
Multi-hop Reasoning	Integrate information distributed across multiple interactions.	Individual facts are retrieved but not composed into the required answer.
Temporal Update	Identify the value valid at the requested temporal point after revisions.	A superseded value is returned for a current query, or a current value is projected incorrectly into the past.
Conflict Resolution	Resolve competing observations using attribution, evidence, and temporal validity.	Competing statements are retrieved but the system fails to identify which one should govern the requested state.
Abstention	Recognize when available evidence does not support a reliable user state.	Missing, hypothetical, or insufficient evidence is converted into a confident answer.

C.2 FINE-GRAINED TASK DISTRIBUTION

Table 18 reports their distribution across interaction scales.

D EXAMPLE LONGITUDINAL TRAJECTORY

Table 19 illustrates the structure of a simplified trajectory in which user information changes over time.

The example illustrates three properties. First, historical information remains recoverable after being superseded. Second, unrelated interactions may intervene between evidence and evaluation. Third, planning requires multiple pieces of currently valid information to be jointly translated into a decision.

E PLANNING AND REPLANNING CONSTRUCTION DETAILS

E.1 PLANNING TASK CONSTRUCTION

Each planning instance combines three information sources:

$$\mathcal{I}_{\text{plan}} = (S_q, R_q, \mathcal{B}) \,, \tag{11}$$

where S_q is the privileged query-time user state used to construct evaluation criteria, R_q is the current request visible to the evaluated system, and $\mathcal{B}$ is the shared structured travel environment.

The evaluated system does *not* receive S_q directly. Relevant user information must be recovered from its own memory of preceding interactions.

Table 17: Complete QA task taxonomy used in the released benchmark. Individual instances may exhibit multiple sources of difficulty, but are assigned a primary category according to the capability targeted during construction.

Task Type	Primary Capability	Description / Example
Simple Retrieval	Direct memory retrieval	Retrieve an explicitly stated user fact, preference, constraint, or trip-related detail from the observable dialogue history. For example, a task may ask for a previously stated transportation preference.
Implicit Memory	Implicit state inference	Recover a user preference or constraint that is implied by prior interactions rather than expressed as a direct declarative fact. This tests whether the memory system can preserve information whose relevance must be inferred from context.
Multi-hop	Evidence integration	Combine information distributed across multiple dialogue turns or events to derive the answer. No single supporting turn is necessarily sufficient on its own, requiring retrieval and integration of multiple pieces of evidence.
Temporal Update	Temporal state tracking	Identify the user state that is valid at the query time after preferences, constraints, goals, or contextual information have changed. Earlier states may remain relevant for historical queries but should not override a later valid update.
Conflict Resolution	Conflict-aware state resolution	Resolve competing or contradictory pieces of information and determine which state is valid for the queried user and time point. This includes separating superseded information from currently actionable information.
Abstention	Evidence-aware abstention	Recognize when the observable dialogue prefix does not provide sufficient support for the requested information and avoid producing an unsupported user-specific claim.

Table 18: Distribution of QA task types across cumulative interaction scales. The four scales are nested prefixes of the same user trajectories; therefore, counts across columns should not be summed.

Task Type	8K	16K	32K	64K
Simple Retrieval	16	32	64	128
Implicit Memory	24	48	96	192
Multi-hop	32	64	128	256
Temporal Update	40	80	160	320
Conflict Resolution	32	64	128	256
Abstention	16	32	64	128
Total QA	**160**	**320**	**640**	**1,280**

E.2 Structured Travel Environment and Database Schema

Planning and replanning tasks operate over a shared structured travel environment. All evaluated memory systems receive the same database for a given task, ensuring that differences in planning performance arise from memory use rather than differences in access to environmental information. The database contains three primary entity types: points of interest (POIs), hotels, and restaurants.

POI records describe properties required for itinerary construction and feasibility checking, including location, category, opening hours, admission price, recommended visit duration, and suitability-related attributes such as indoor availability, elderly suitability, child suitability, and thrill-related characteristics. Hotel records contain location, price, room or bed information, and accommodation-related attributes. Restaurant records contain location, price, cuisine type, opening hours, and representative dishes.

Table 20 summarizes the major database attributes and their roles in planning evaluation.

Table 19: Illustrative longitudinal trajectory. Only the *Dialogue / Task* column is visible to the evaluated memory system; event and state columns are privileged benchmark annotations.

Stage	Dialogue / Task	Event Annotation	Ground-Truth State
t_1	"I usually choose inexpensive hotels when I travel."	Preference added	Hotel preference: budget-oriented
t_2	"My father may travel with us, so we should avoid too much walking."	Constraint added	Walking intensity: low
t_3	"I recently started learning how to bake sourdough bread at home."	External distractor	No state change
t_4	"Recently I care more about having a swimming pool than choosing the cheapest hotel."	Preference revised	Hotel preference: pool $>$ lowest price
q_1	**QA:** "What matters most to the user when choosing a hotel now?"	Current-state QA	Pool availability
q_2	**QA:** "What did the user previously prioritize when choosing hotels?"	Historical-state QA	Low price
q_3	**Planning:** "Plan a multi-day family trip."	Planning task	Current hotel preference + walking constraint

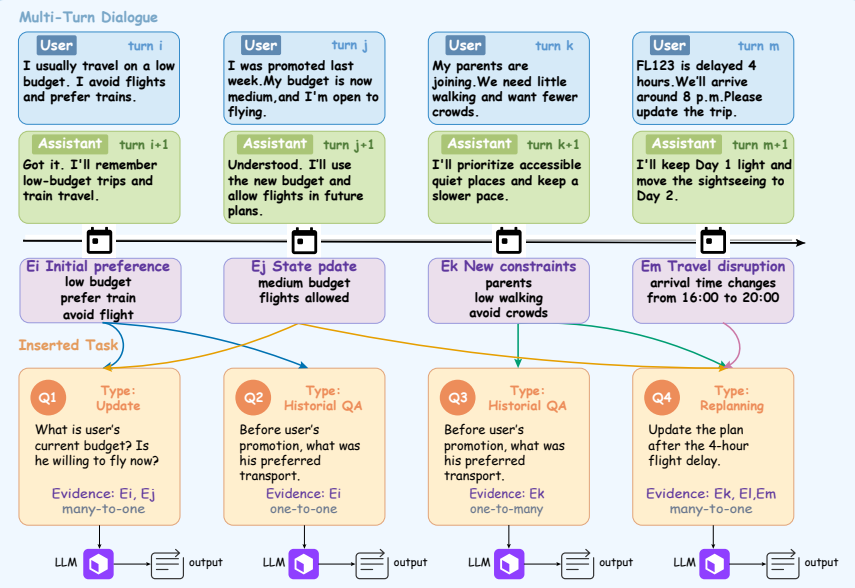

Figure 4: Illustrative longitudinal trajectory with evolving user states, inserted memory tasks, historical evidence, and replanning. User dialogue is observed chronologically, while state annotations and evidence links are used only for task construction and evaluation.

Use in deterministic evaluation. The structured database is used to evaluate constraints that can be verified without subjective language-model judgment. For example, opening hours and recommended duration support temporal-feasibility checks; price information supports budget-related checks; and structured suitability attributes support constraints involving elderly travelers, children, indoor activities, or activity characteristics. Restaurant and hotel attributes similarly support checks of availability, price, cuisine, location, and accommodation requirements.

Not every personalization criterion is reducible to a database lookup. Constraints derived from longitudinal user preferences, companion information, or contextual requirements are first represented as atomic evaluation criteria with explicit provenance. Database-grounded criteria are evaluated against the structured environment, while criteria requiring semantic interpretation are handled by the corresponding evaluation procedure described in Appendix H.3.

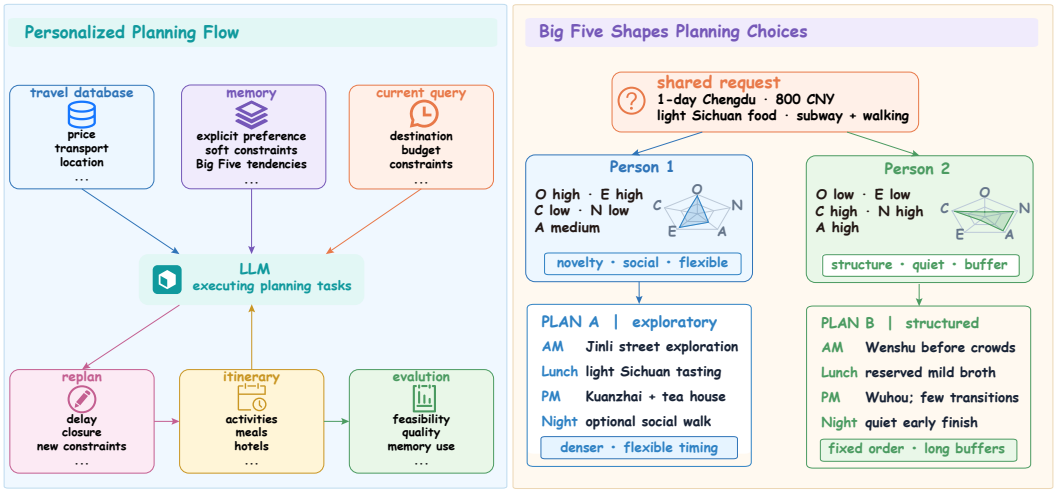

Figure 5: Personalized planning flow and personality-conditioned planning choices. The planner combines the structured travel database, retrieved memory, and the current request to produce and evaluate an itinerary. Different Big Five profiles can lead to different valid plans under the same request.

Table 20: Structured travel-database attributes used by planning and replanning tasks. The same environment is provided to all evaluated methods. Evaluation-relevant fields support deterministic checks whenever the corresponding constraint can be resolved directly from structured database information.

Entity	Representative structured attributes	Evaluation role
POI	Name, location, category, opening hours, price, recommended duration, indoor/outdoor attributes, elderly suitability, child suitability, and thrill-related attributes	Checks attraction availability, temporal feasibility, cost, visit duration, and compatibility with user- or companion-specific activity constraints.
Hotel	Name, location, price, room information, bed information, and accommodation features	Checks lodging selection, budget compatibility, location requirements, and explicit accommodation constraints.
Restaurant	Name, location, price, cuisine type, opening hours, and representative dishes	Checks meal availability, budget compatibility, cuisine or food preferences, and restaurant-related constraints.

The complete released database schema, database files, and deterministic evaluation implementation are provided with the benchmark code at `https://anonymous.4open.science/r/LifeMemBench-5B33/`.

E.3 Atomic Constraint Construction

Each planning instance is associated with a set of atomic criteria

$$C_q = \{c_1, c_2, \ldots, c_m\}, \tag{12}$$

rather than a single reference itinerary. This reflects the fact that multiple itineraries may be valid for the same user.

We distinguish three high-level evaluation dimensions:

Table 21: Constraint provenance used in planning evaluation.

Provenance	Interpretation
current.request	Directly available in the current planning request.
memory.derived	Requires user information introduced in earlier interactions.
companion.derived	Depends on previously introduced information about a travel companion.
database	Determined by attributes in the structured travel environment.
derived	Requires composition of multiple information sources.

- **Mandatory constraints:** requirements that should not be violated, including explicit budgets, temporal restrictions, mandatory activities, prohibited options, and hard mobility requirements;
- **Personalization constraints:** user-specific preferences, habits, companion requirements, transportation preferences, accommodation preferences, dietary requirements, and other criteria derived from longitudinal interaction history; and
- **Feasibility constraints:** requirements grounded in the structured environment, including opening times, prices, availability, geographic compatibility, and other executable planning conditions.

E.4 Constraint Provenance

Not every satisfied criterion constitutes evidence that long-term memory has been used. Some requirements are directly available in the current planning request. We therefore annotate the provenance of every atomic criterion.

Table 21 summarizes the provenance categories used for planning constraints.

Analyses of the transition from memory to action focus on `memory_derived` criteria and history-dependent `companion_derived` criteria.

Planning-constraint composition. Planning supervision is constructed from three explicitly represented sources: the current trip request, longitudinal memory constraints, and structured database constraints. The released annotations preserve these sources separately through `trip_request`, `memory_constraints`, and `database_constraints`. Evaluation criteria are subsequently derived from these annotations as atomic hard constraints, personalization requirements, and avoidance constraints.

We do not report a five-way criterion-provenance distribution because provenance labels were not stored at the individual atomic-criterion level in the released construction artifacts. Rather than retrospectively assigning potentially ambiguous provenance labels, we retain the explicit source annotations provided by the construction pipeline.

E.5 Planning–QA Linkage Schema

To study whether recovered memory is translated into downstream action, we construct constraint-level links between diagnostic QA targets and planning requirements derived from the same underlying user-state evidence.

Each linkage record contains:

- `system`;
- `context_length`;
- `sample_id`;
- `episode_id`;
- `planning_id`;
- `parent_planning_id`;
- `constraint_id`;
- `constraint_category`;

- `constraint_source`;
- `polarity`;
- supporting event and dialogue-turn identifiers;
- `target_state`;
- linked QA identifiers;
- QA outcome; and
- planning-constraint outcome.

Only constraints whose satisfaction genuinely depends on historical memory are used for direct memory–action linkage analysis.

E.6 QA–Planning Pairing Protocol

We first match QA targets and planning constraints using shared event identifiers or supporting dialogue evidence. Candidate pairs must additionally agree in trajectory or episode, relevant entity, temporal target where applicable, and the state transition involved.

Pairs without reliable structural support are marked *unlinked*; we do not force associations using text similarity alone.

Constraint outcomes distinguish *satisfied*, *partially satisfied*, *violated*, and *omitted*.

The resulting paired examples support a direct analysis of four cases:

	Constraint Satisfied	Constraint Failed
QA Correct	✓	Utilization Failure
QA Incorrect	Indirect/Accidental Success	Memory Failure

(13)

E.7 Planning Constraint Evaluation

Each criterion is represented by a structured record containing at least

$$c_j = \langle \text{type}, \text{description}, \text{source}, \text{verification rule} \rangle . \tag{14}$$

Whenever possible, criteria are evaluated deterministically using the structured travel environment and the generated itinerary. Examples include price limits, opening-hour compatibility, mandatory attractions, availability, and other explicitly represented conditions.

Criteria requiring semantic interpretation are converted into explicit criterion-level rubrics and evaluated using a structured LLM judge. The judge evaluates individual requirements rather than assigning an unconstrained holistic preference score.

Reference itineraries are retained as auxiliary evaluation context where useful but are not treated as unique correct solutions.

E.8 Replanning Task Construction

Each replanning instance begins with an existing itinerary and introduces a new circumstance that invalidates part of the previous solution. The change may originate from the environment or from newly introduced user or companion information.

Replanning is evaluated under two complementary requirements:

$$\text{Replanning Success} = \text{Adapt Invalidated Requirements} + \text{Preserve Still-Valid Requirements}. \tag{15}$$

The first component measures whether the newly introduced circumstance is correctly propagated into the revised itinerary. The second measures whether the system unnecessarily discards previously accumulated personalization that remains applicable.

E.9 Illustrative End-to-End Planning and Replanning Example

This section provides an end-to-end example illustrating how a longitudinal user history is converted into planning constraints and how the same memory must subsequently support replanning after a change in user state or the environment. For readability, we show only dialogue turns relevant to the planning instance; unrelated dialogue and injected external distractors are omitted. The evaluated system observes only the chronological dialogue prefix and the current planning request, while user-state annotations, criterion provenance, and evaluation results are hidden and used only for evaluation.

1. Initial planning request. At query time, the user requests a personalized itinerary:

> **User:** "Please plan a two-day trip for us. We would like a fairly relaxed schedule, and please choose suitable attractions, restaurants, and accommodation based on what I have told you before."

The current request alone does not restate all personalization constraints. The planner must therefore recover relevant information from the preceding longitudinal interaction history.

2. Relevant longitudinal evidence. The following excerpts illustrate information accumulated before the planning query:

> **Earlier interaction t_1:** "I usually prefer taking public transportation when travelling rather than using taxis for every trip."
>
> **Earlier interaction t_2:** "I like historical and cultural attractions more than thrill-oriented activities."
>
> **Earlier interaction t_3:** "My mother will be travelling with me this time, so I would rather avoid places that require a lot of walking or strenuous activities."
>
> **Earlier interaction t_4:** "For this trip I would like to keep the overall spending moderate."
>
> **Later update t_5:** "We do not need to avoid outdoor places completely. My mother is fine with easy outdoor activities as long as they are not physically demanding."

The final statement refines the earlier mobility-related constraint rather than eliminating it. A memory system must therefore preserve the historical evidence while recovering the state that is valid at the planning query.

3. Query-time user-state constraints. From the observable prefix, the benchmark associates the planning task with the following query-time constraints:

State dimension	Valid constraint at query time
Transportation	Prefer public transportation over unnecessary taxi use.
Activity preference	Prefer historical and cultural attractions.
Companion mobility	Avoid strenuous or walking-intensive activities because an older companion is travelling with the user.
Outdoor activity	Outdoor activities are acceptable when physically easy.
Budget	Prefer moderate overall spending.
Pace	Prefer a relatively relaxed itinerary.

4. Atomic criteria and provenance. The planning evaluator converts the request, longitudinal state, companion information, and structured travel environment into atomic criteria. A representative subset is shown below.

Table 22 shows the atomic criteria for this example.

This representation allows evaluation to distinguish criteria originating from the current request from those that require longitudinal memory, companion-specific information, or structured environmental information.

Table 22: Representative atomic criteria for the initial planning instance.

ID	Criterion	Source	Evaluation
C1	Construct a feasible two-day itinerary.	Current request	Hard / feasibility
C2	Prefer public transportation when practical.	Longitudinal memory	Personalization
C3	Prioritize historical or cultural attractions.	Longitudinal memory	Personalization
C4	Avoid strenuous or walking-intensive activities.	Companion information	Hard / personalization
C5	Easy outdoor activities remain acceptable.	Longitudinal update	Personalization
C6	Keep selected activities and services compatible with a moderate budget.	Longitudinal memory	Hard / personalization
C7	Avoid an unnecessarily dense schedule.	Current request + memory	Personalization
C8	Visit selected POIs only during their available opening hours.	Structured database	Feasibility

5. Generated initial plan. A successful planner may produce an itinerary of the following form:

> **Day 1.** Travel by public transportation to a historical museum in the morning, followed by lunch at a moderately priced nearby restaurant. Visit an accessible cultural site in the afternoon and return to the hotel before dinner.
>
> **Day 2.** Visit an easy-access heritage attraction in the morning. After lunch, choose a low-intensity outdoor cultural area with sufficient rest time, followed by a nearby dinner before returning to the hotel.

The actual benchmark output contains concrete entities from the shared structured travel database; the compact presentation above emphasizes the memory-to-action dependencies relevant to this example.

6. Constraint-level evaluation. Rather than assigning only a single holistic score, the evaluator checks the generated itinerary against individual criteria:

ID	Evaluation result	Satisfied?
C1	Two-day structure and schedule are feasible	✓
C2	Public transportation is used where practical	✓
C3	Cultural/historical attractions are prioritized	✓
C4	No strenuous activity is selected	✓
C5	Selected outdoor activity is low intensity	✓
C6	Selections remain compatible with the budget constraint	✓
C7	Daily schedule leaves adequate transition/rest time	✓
C8	Visits fall within database opening hours	✓

These atomic outcomes contribute to the reported hard-constraint, personalization-coverage, and feasibility measurements.

7. Subsequent state/environment update. After the initial itinerary has been produced, a new event changes the conditions under which the trip must be executed:

> **User:** "My mother has been feeling tired recently, so for the second day I want to avoid outdoor activities and keep walking to a minimum."

This update supersedes the earlier state in which easy outdoor activities were acceptable. Historical evidence remains valid as history, but should no longer determine the corresponding Day-2 decision.

8. Replanning request. The system subsequently receives:

> **User:** "Please update the itinerary based on this change while keeping the parts that still work."

The replanning task therefore requires both *state resolution* and *minimal plan revision*: the system must identify which parts of the existing itinerary have become invalid without unnecessarily replacing unaffected decisions.

9. Required-change annotations. The benchmark specifies which parts of the previous plan must change:

Required change	Reason
Replace Day-2 outdoor activity	The new companion-mobility constraint invalidates the previously acceptable outdoor activity.
Reduce walking intensity	The latest state explicitly requires minimal walking.
Preserve Day-1 activities	No new information invalidates the feasible Day-1 itinerary.
Preserve transportation preference	The preference for public transportation remains valid.
Preserve budget constraint	No budget update has occurred.

These annotations distinguish necessary adaptations from unnecessary changes to portions of the original plan that remain valid.

10. Revised plan and evaluation. A compliant revised itinerary replaces the affected Day-2 activity with an accessible indoor cultural attraction while retaining the unaffected parts of the original itinerary:

> **Revised Day 2.** Visit an accessible indoor cultural attraction in the morning, using public transportation and minimizing walking between stops. After lunch, schedule an indoor museum or exhibition with sufficient rest time. Retain the original moderately priced restaurant and transportation choices where they remain feasible.

The revised plan is evaluated against both persistent and newly introduced constraints. In particular, the evaluator checks that the invalidated outdoor activity has been removed, walking demand has been reduced, unaffected valid choices have been preserved where possible, and the original transportation, budget, temporal, and feasibility constraints remain satisfied.

This example illustrates the distinction targeted by LifeMemBench: successful longitudinal memory requires more than retrieving a previously stated fact. The system must determine which information is valid at the current query time, ground planning decisions in that state, detect when a later update invalidates part of an existing plan, and propagate the change to downstream actions without discarding still-valid information.

F Complete CoMem Implementation Details

F.1 Interaction and Event Memory

Interaction memory. Interaction memory $\mathcal{C}$ retains source-level conversational units. Its primary role is evidence preservation: compression or state resolution should not permanently remove the original dialogue required by later historical or evidence-sensitive queries.

Event memory. Event memory $\mathcal{E}$ stores compact state-relevant observations extracted from interactions. Each event retains its content together with temporal and source information required to trace it back to the corresponding interaction.

Conceptually, an event is represented as

$$e_i = (s_i, o_i, m_i, t_i, r_i)\,, \tag{16}$$

where s_i denotes the subject, o_i the inferred operation, m_i the memory content, t_i the timestamp, and r_i the source reference.

F.2 Derived Current-State Snapshot

For planning-oriented tasks, CoMem derives a compact snapshot from eligible event memories:

$$\hat{S}_t = F_{\text{snap}}\left(\mathcal{E}_{<t}\right). \tag{17}$$

The snapshot is not treated as an independent source of truth. It is a derived view of event history intended to expose currently applicable user information to a small downstream backbone. Source events and original interactions remain available, preventing state compression from destructively removing historical evidence.

When multiple events concern the same user attribute, snapshot construction uses temporal order, update operations, attribution, and validity information to determine the currently actionable value.

F.3 Selective Write and Noise Gate

Noise Gate. The Noise Gate is a lightweight pre-filter applied before event extraction and state writing. Its purpose is not to determine the final user state, but to prevent clearly irrelevant content from triggering unnecessary memory updates. Importantly, filtering affects only the state-maintenance path: all dialogue turns remain preserved in the raw interaction memory regardless of the gate decision. Thus, an incorrect filtering decision does not delete the original evidence.

The released implementation applies the following rules:

1. **User-state-relevant user turns are accepted.** A user turn is forwarded to the event writer when it may introduce, modify, refine, contradict, invalidate, or contextualize information relevant to the user state. This includes preferences, constraints, goals, plans, budget, pace, dietary requirements, transportation preferences, companion information, and other travel-relevant context.
2. **Clearly external distractors are rejected from state writing.** Turns identified as unrelated external content, such as injected programming questions, mathematical exercises, reading-comprehension material, or unrelated general conversations, are not forwarded as candidate user-state updates. They remain available in raw interaction memory but do not modify the derived state.
3. **State-relevant assistant or entity information is retained when it can affect subsequent user state.** Assistant turns are not treated as user preferences by default. However, information introduced by the assistant or another entity is retained when it provides context necessary to interpret a subsequent user response or state transition. Attribution is preserved so that assistant- or companion-originated information is not incorrectly written as a first-person user preference.
4. **Ambiguous but potentially relevant turns are delegated to the event writer.** The gate is intentionally conservative. If a turn cannot be confidently classified as irrelevant but may contain a state-changing signal, it is forwarded to the event writer rather than discarded. The event writer subsequently determines whether an event should be created and whether that event changes the derived current-state snapshot.

Table 23 gives representative examples of these decisions.

This asymmetric design favors false positives over false negatives at the gate: potentially relevant information can still be rejected by the downstream event writer, whereas prematurely removing a relevant turn would prevent it from contributing to state maintenance. Because the original interaction is preserved independently, the Noise Gate controls the state-writing path without serving as an irreversible deletion mechanism.

F.4 Memory Writer

Memory-writing prompt. After the Noise Gate, potentially state-relevant interactions are passed to the memory writer. The writer converts the local interaction and relevant memory context into a structured event while explicitly separating state-changing information from historical, hypothetical, and third-party content. The released implementation uses the following operation space: ADD, UPDATE, REMOVE, and NONE.

Table 23: Representative Noise Gate decisions. "Rejected" means excluded from the state-writing path, not removed from raw interaction memory.

Case	Representative turn	Noise Gate action
Accepted user turn	"I would rather use public transportation than take taxis during this trip."	Forward to event writer as a candidate transportation preference.
Accepted state update	"Actually, I am fine with outdoor attractions now, as long as they do not require too much walking."	Forward to event writer as a candidate refinement/update of the current state.
Rejected external distractor	"Write a Python function that returns the factorial of an integer."	Do not trigger state writing; retain only in raw interaction memory.
Rejected external distractor	"If a train travels 120 km in two hours, what is its average speed?"	Do not trigger state writing; retain only in raw interaction memory.
Retained contextual information	*Assistant:* "The attraction involves approximately two hours of walking." *User:* "Then that probably will not work for my mother."	Retain the contextual turn and forward the user response with its attribution and context to the event writer.
Third-party information	"My mother prefers places where she can sit down and rest frequently."	Forward as companion-related information; preserve subject attribution rather than writing it as the user's own preference.
Ambiguous turn	"That sounds better for this trip."	Do not force a state update at the gate; forward with local context to the event writer for interpretation.

Memory Writer Instruction

Given the current dialogue turn, its local context, and relevant existing memory, determine whether the interaction introduces or changes information that should be represented in memory. Preserve the subject, temporal scope, and relationship between previous and new information. Do not treat every mentioned fact as a change to the current user state.

Operations

- **ADD.** Use ADD when the interaction introduces previously unrecorded information about the user or another explicitly identified subject that may be relevant to future interactions. *Example:* "I usually prefer travelling by metro." ADD : user transportation preference = metro/public transit.
- **UPDATE.** Use UPDATE when newly observed information modifies, refines, corrects, or replaces an existing memory. Preserve both the previous and new values rather than silently overwriting the historical state.
  Example:
  Existing memory: user prefers a packed itinerary.
  New turn: "For this trip, I would rather take things slowly."
  UPDATE : pace = packed $\rightarrow$ relaxed.
- **REMOVE.** Use REMOVE when the interaction explicitly invalidates, withdraws, or indicates that a previously stored constraint or preference no longer applies. Preserve the removed information as historical evidence while marking it as no longer actionable in the current state.
  Example:
  Existing memory: user avoids outdoor attractions.
  New turn: "That restriction no longer applies."

REMOVE : current outdoor-avoidance constraint; retain the former constraint in history.

- **NONE.** Use NONE when the interaction does not justify a memory-state change. This includes irrelevant external content, acknowledgements, unsupported interpretations, and contextual statements that do not introduce or modify a persistent state.
  Example:
  "Thanks, that sounds good."
  NONE.

Attribution rules

Information must be assigned to the subject who actually expresses or owns the state. A preference concerning a companion must not be converted into a first-person user preference.

Example:

"My mother prefers restaurants with mild food."

subject = mother/companion; attribute = food preference; value = mild food.

It must *not* be stored as "the user prefers mild food."

Hypothetical and conditional information

Hypothetical, conditional, or imagined statements must not directly overwrite the actionable current state. Preserve their non-factual status when they may be useful as historical evidence.

Example:

"If I travelled alone, I would probably choose a much faster itinerary."

hypothetical = true; do not replace the user's current pace preference.

Historical statements

Statements explicitly referring to a previous state should be preserved as historical evidence but should not automatically become the current state.

Example:

"I used to prefer very cheap hotels, but that was several years ago."

historical = true; retain the former preference without treating it as the current hotel-budget preference.

Old–new relation preservation

For an UPDATE or REMOVE, preserve the relation between the previous memory and the newly observed information. Do not discard the superseded value. The resulting event should contain sufficient information to reconstruct both the state before the event and the state after the event.

Example:

Existing memory: "The user avoids outdoor activities."

New turn: "Outdoor places are fine now as long as they do not involve much walking."

`operation` = UPDATE,
`old value` = avoid outdoor activities,
`new value` = allow low-intensity outdoor activities,
`relation` = refinement/replacement.

The old value remains historical evidence while the new value becomes actionable.

General principles

1. Do not infer a persistent preference from weak or unrelated evidence.
2. Preserve user-versus-companion attribution.
3. Distinguish factual, hypothetical, conditional, and historical statements.
4. Do not overwrite old information when an update occurs; preserve the old–new relation.
5. When the evidence is insufficient to justify a state change, prefer NONE.

The writer output is stored in event memory rather than directly overwriting the user profile. The current-state snapshot is subsequently derived from these attributed and temporally ordered events. This separation allows CoMem to maintain an actionable current state while retaining superseded, historical, hypothetical, and third-party information for later temporal or historical queries.

F.5 TEMPORAL GUARD

The Temporal Guard constrains retrieval according to the temporal scope of the task.

Eligibility filtering. Memories occurring after the current task position are always excluded. For historical queries with an explicit target time t_q, events occurring after the target time are removed from the historical evidence view:

$$\mathcal{E}_{\leq t_q} = \{e_i \in \mathcal{E} : t(e_i) \leq t_q\}. \tag{18}$$

Query interpretation. Queries are assigned one of three coarse temporal modes: CURRENT, HISTORICAL, or CHANGE. The inferred mode modifies temporal relevance signals and the interpretation of retrieved evidence without requiring a separate temporal reasoning agent.

The guard does not delete historical memories; it controls which memories are eligible for a particular task.

Released rule definitions. The released implementation uses lightweight deterministic rules to infer the temporal mode of a query before retrieval. Query text is normalized to lower case and matched against temporal expressions associated with three coarse modes: CURRENT, HISTORICAL, and CHANGE.

- **CURRENT.** Queries referring to the user's presently valid state are assigned to CURRENT. Representative cues include `current`, `currently`, `now`, `right now`, `latest`, `still`, and constructions such as `what does the user prefer` or `what is the user's preference`. When no explicit historical or change cue is detected, CURRENT is used as the default mode.
- **HISTORICAL.** Queries requesting the state at an earlier point are assigned to HISTORICAL. Representative cues include `previously`, `before`, `earlier`, `at the time`, `used to`, `originally`, `back then`, and explicit temporal expressions such as `before <date>` or `as of <date>`. When a target time t_q is available from the task metadata, retrieval is restricted to memories satisfying $t(e_i) \leq t_q$.
- **CHANGE.** Queries explicitly asking how a state evolved are assigned to CHANGE. Representative cues include `change`, `changed`, `update`, `updated`, `from ... to ...`, `how has`, `what changed`, and `difference between`. For these queries, both the earlier and later state-transition evidence is retained when it occurs within the observable temporal boundary.

The keyword rules determine only the coarse query mode; they do not directly produce an answer or modify stored memories. Temporal eligibility is enforced independently of lexical matching. For every task at dialogue position q, any memory originating after the task position is excluded. For a historical task with target time t_q, memories after t_q are additionally excluded. Consequently, the eligible evidence set is

$$\mathcal{E}_{\text{eligible}}(q) = \begin{cases} \{e_i \mid t(e_i) \leq t_q\}, & \text{HISTORICAL}, \\ \{e_i \mid \text{pos}(e_i) < q\}, & \text{CURRENT or CHANGE}. \end{cases} \tag{19}$$

If multiple temporal cues occur in the same query, explicit historical target-time information takes precedence over generic current-state cues, while explicit change-oriented constructions select CHANGE when the task asks for a transition rather than a state at a single time point. Memories filtered from a particular retrieval view remain unchanged in the underlying event and interaction stores.

F.6 HYBRID RETRIEVAL

Final retrieval score. Temporal eligibility is enforced as a hard constraint before hybrid relevance scoring. Let $g_{\text{temp}}(m, q) \in \{0, 1\}$ indicate whether memory m is temporally eligible for query q under the Temporal Guard defined in Section F.5. Memories with $g_{\text{temp}}(m, q) = 0$ are removed from the candidate set and therefore cannot be recovered solely because of high semantic similarity.

For an eligible memory, the released hybrid retrieval score is

$$R(m,q) = \lambda_s R_{\text{sem}}(m,q) + \lambda_l R_{\text{lex}}(m,q) + \lambda_t R_{\text{temp}}(m,q), \qquad g_{\text{temp}}(m,q) = 1. \tag{20}$$

Here R_{sem} is the cosine similarity between the query and memory embeddings, R_{lex} measures lexical overlap between the query and memory text, and R_{temp} provides a lightweight temporal compatibility signal within the temporally eligible candidate set. Equivalently, the complete retrieval rule can be written as

$$R^*(m,q) = \begin{cases} \lambda_s R_{\text{sem}}(m,q) + \lambda_l R_{\text{lex}}(m,q) + \lambda_t R_{\text{temp}}(m,q), & g_{\text{temp}}(m,q) = 1, \\ -\infty, & g_{\text{temp}}(m,q) = 0. \end{cases} \tag{21}$$

The eligible candidates are ranked by $R^*(m,q)$, and the top-k memories are returned as retrieval evidence. No LLM-based reranker is used. Entity and attribution metadata are retained with the retrieved memories for downstream state interpretation, rather than introduced as independent additive score terms. Similarly, temporal eligibility is not implemented as a soft recency bonus: future or otherwise temporally invalid memories are excluded before ranking.

This design deliberately separates *eligibility* from *relevance*. The Temporal Guard determines whether a memory may participate in retrieval, whereas hybrid scoring determines which of the eligible memories are most relevant to the query. Consequently, a highly similar but temporally invalid memory cannot outrank valid historical evidence.

F.7 Task-Specific Context Construction

For QA, the context emphasizes source interactions and, when needed, event-linked evidence:

$$X_{\text{QA}} = Q \oplus R_C \oplus R_E. \tag{22}$$

For planning and replanning, the context emphasizes currently actionable state together with relevant event evidence and the structured travel environment:

$$X_{\text{plan}} = Q \oplus \hat{S}_t \oplus R_E \oplus \mathcal{B}_t. \tag{23}$$

F.8 Retrieval and Generation Hyperparameters

Table 24 reports the configuration used for all reported CoMem experiments. Unless otherwise stated, the same configuration is shared across the 8K, 16K, 32K, and 64K interaction scales.

CoMem first retrieves a small interaction-level candidate pool and then combines semantic, lexical, and temporal relevance signals after temporal eligibility filtering. The final interaction evidence set contains at most five entries. Event retrieval uses a conservative rescue mechanism that can add at most one event when relevant state-transition evidence is not already represented in the selected interaction evidence. Planning uses a slightly larger event budget because a single itinerary may depend on multiple preferences, constraints, and companion-specific states.

Additional retrieval details. Semantic relevance is computed using the shared Qwen3-Embedding-0.6B embedding model. Embedding similarity is combined with lightweight lexical matching and temporal compatibility as described in Section F.6. Temporal eligibility is applied as a hard filter before final ranking, so temporally invalid memories cannot be rescued by a high semantic or lexical score.

The released CoMem configuration does not employ an LLM-based reranker. Likewise, no separate BM25 retrieval stage is required by the reported configuration; lexical relevance is computed directly within the lightweight hybrid scorer. Entity and attribution information is retained as structured memory metadata for downstream interpretation rather than introduced as independent additive retrieval bonuses. Consequently, there are no additional BM25 hyperparameters or entity-matching bonus coefficients to tune.

Table 24: Default retrieval and generation configuration of CoMem used in all reported experiments.

Parameter	Value
Interaction candidate pool	12
Final interaction Top-K	5
Event rescue limit	1
Planning event Top-K	8
Planning snapshot entries	6
Semantic weight λ_s	0.55
Lexical weight λ_l	0.40
Temporal weight λ_t	0.05
QA hybrid semantic weight α	0.80
Memory extraction max tokens	512
QA generation max tokens	128
Planning generation max tokens	4096

Table 25: Implementation details of evaluated memory systems. All methods use the same Qwen3.5-4B generation backbone; method-specific memory construction and retrieval mechanisms are preserved whenever supported by the original implementation.

Method	Memory Representation	Retrieval Configuration
Mem0	Extracted long-term memory items	Native semantic memory search; retrieved memories are supplied to the shared generation backbone.
MemoryBank	Summarized long-term memories with structured historical records	Embedding-based retrieval with Top-$K = 12$.
A-MEM	Agentic memory notes with dynamically organized memory attributes and relationships	Native A-MEM retrieval over constructed memory notes.
MIRIX	Multi-component memory representation separating complementary memory types	Native retrieval over the maintained MIRIX memory components.
MemBuilder	Incrementally constructed structured memory representation	Native retrieval over the constructed memory store.
Letta	Persistent agent memory with archival and working-memory components	Native Letta memory retrieval under the shared evaluation interface.
CoMem	Interaction memory + event memory + derived current-state snapshot	Task-adaptive hybrid retrieval with temporal eligibility filtering; interaction Top-$K = 5$ and planning event Top-$K = 8$.

The event-rescue mechanism is bounded by the rescue limit reported above and is applied only after the primary interaction retrieval stage. Retrieved evidence is clipped to the corresponding generation context budget before being passed to the Qwen3.5-4B generation backbone. All reported experiments use the same retrieval configuration without scale-specific hyperparameter tuning.

G Baseline Implementation Details

For fair comparison, we preserve each system's native memory representation and update mechanism whenever possible while standardizing the downstream generation backbone and evaluation protocol. Qwen3.5-4B is used as the shared generation backbone; retrieval-based methods retain their corresponding retrieval or embedding components.

Table 25 summarizes the baseline memory representations and update rules.

For each method, the released configuration documents:

- source repository and commit hash;
- modifications required for Qwen3.5-4B;

- memory-writing prompt or update policy;
- embedding model, if applicable;
- retrieval Top-K;
- memory limits or eviction policy;
- generation configuration;
- method-specific fallback behavior; and
- deviations from the original implementation.

H Additional Evaluation Metrics

H.1 Abstention Diagnostics

The Answerable Refusal Rate is

$$\mathrm{ARR} = \frac{N_{\text{refused answerable}}}{N_{\text{answerable}}}. \tag{24}$$

Reporting this measure alongside abstention performance avoids rewarding systems that improve apparent abstention simply by refusing more aggressively.

H.1.1 Stale-State Adoption

Stale-State Adoption Rate measures how often a response relies on information that was previously valid but had been superseded before the query:

$$\mathrm{SAR} = \frac{N_{\text{responses adopting stale state}}}{N_{\text{stale eligible cases}}}. \tag{25}$$

Lower is better.

H.2 Update Omission

Update Omission Rate measures how often a relevant update that should affect the answer is not reflected in the generated response:

$$\mathrm{UOR} = \frac{N_{\text{omitted relevant updates}}}{N_{\text{eligible update cases}}}. \tag{26}$$

Relation to QA-UMR. Update Omission Rate (UOR) and QA Unsupported/Outdated Memory Rate (QA-UMR) capture complementary failure modes and are computed over different error conditions. UOR measures whether information introduced by a relevant state update is *missing* from a response when that update should affect the answer:

$$\mathrm{UOR} = \frac{\sum_{i \in \mathcal{Q}_{\mathrm{upd}}} \mathbb{I}[\text{required update is not reflected in } \hat{y}_i]}{|\mathcal{Q}_{\mathrm{upd}}|}, \tag{27}$$

where $\mathcal{Q}_{\mathrm{upd}}$ contains QA instances for which at least one preceding update is required to determine the correct query-time state.

In contrast, QA-UMR measures whether a response actively relies on memory content that is unsupported or no longer valid at the query time:

$$\text{QA-UMR} = \frac{\sum_{i \in \mathcal{Q}_{\mathrm{mem}}} \mathbb{I}[\hat{y}_i \text{ adopts unsupported or outdated memory}]}{|\mathcal{Q}_{\mathrm{mem}}|}, \tag{28}$$

where $\mathcal{Q}_{\mathrm{mem}}$ denotes QA instances for which the response can be checked against the annotated query-time memory state.

The distinction is between *omission* and *incorrect adoption*. For example, suppose an earlier preference for taxis is later updated to a preference for public transportation. A response that fails to mention the updated transportation preference constitutes an update omission. A response that explicitly answers "taxi" adopts the superseded state and is therefore counted by QA-UMR. The two errors can co-occur, but neither definition requires the other.

We report QA-UMR in the main dynamic-state results because it directly measures contamination of generated answers by invalid memory. UOR is retained as a complementary diagnostic for determining whether failures arise from missing a relevant update rather than explicitly adopting stale information. For both metrics, lower values indicate better performance.

H.3 Planning and Replanning Metrics

Planning evaluation decomposes each instance into atomic criteria $\mathcal{C}_i = \{c_{i1}, \ldots, c_{in_i}\}$. Each criterion records its constraint type and evaluation target, allowing the evaluator to distinguish hard feasibility requirements from personalization requirements. Let $s(c, \hat{p}) \in \{0, 1\}$ denote whether generated plan $\hat{p}$ satisfies criterion c.

Hard Constraint Rate (HCR). Let $\mathcal{H}_i \subseteq \mathcal{C}_i$ denote the hard constraints of instance i. HCR measures the fraction satisfied by the generated plan:

$$\mathrm{HCR}_i = \frac{1}{|\mathcal{H}_i|} \sum_{c \in \mathcal{H}_i} s(c, \hat{p}_i). \tag{29}$$

Hard constraints include requirements whose violation makes a plan incompatible with the request or user state, such as explicit budget, mobility, dietary, temporal, or other mandatory restrictions.

Personalization Coverage (PC). Let $\mathcal{P}_i \subseteq \mathcal{C}_i$ denote personalization criteria derived from user preferences, longitudinal memory, and relevant companion information. PC measures how many of these requirements are reflected in the plan:

$$\mathrm{PC}_i = \frac{1}{|\mathcal{P}_i|} \sum_{c \in \mathcal{P}_i} s(c, \hat{p}_i). \tag{30}$$

Unlike HCR, PC captures whether remembered information actually influences the generated itinerary even when omitting that preference would not make the plan strictly infeasible.

Feasibility. Let $\mathcal{F}_i$ denote atomic feasibility checks grounded in the structured travel environment. These checks include applicable constraints such as entity availability, opening hours, temporal compatibility, duration, location, and other database-verifiable requirements. We compute

$$\mathrm{Feasibility}_i = \frac{1}{|\mathcal{F}_i|} \sum_{c \in \mathcal{F}_i} s(c, \hat{p}_i). \tag{31}$$

Whenever possible, feasibility is evaluated deterministically against the structured database rather than inferred solely from surface-form plan text.

Initial-planning score. The overall score for an initial planning instance combines hard-constraint satisfaction, personalization coverage, and feasibility:

$$S_i^{\mathrm{plan}} = \frac{1}{3} \left(\mathrm{HCR}_i + \mathrm{PC}_i + \mathrm{Feasibility}_i \right). \tag{32}$$

The reported planning score is the macro-average over the N_{plan} initial planning instances:

$$S^{\mathrm{plan}} = \frac{1}{N_{\mathrm{plan}}} \sum_{i=1}^{N_{\mathrm{plan}}} S_i^{\mathrm{plan}}. \tag{33}$$

Required Change Coverage (RCC). For replanning instance i, let Δ_i denote the set of annotated changes required after the state or environment update. RCC measures the fraction correctly incorporated into the revised plan:

$$\mathrm{RCC}_i = \frac{1}{|\Delta_i|} \sum_{\delta \in \Delta_i} \mathbb{I}[\delta \text{ is correctly reflected in } \hat{p}_i']. \tag{34}$$

This explicitly evaluates whether a system propagates newly introduced information into downstream action.

Replanning score. The revised plan is evaluated against the currently valid hard constraints, personalization criteria, and feasibility checks, together with required change coverage:

$$S_i^{\mathrm{replan}} = \frac{1}{4} \left(\mathrm{HCR}_i' + \mathrm{PC}_i' + \mathrm{Feasibility}_i' + \mathrm{RCC}_i \right), \tag{35}$$

where the primed quantities are recomputed for the revised plan under the post-update state. The reported replanning score is

$$S^{\mathrm{replan}} = \frac{1}{N_{\mathrm{replan}}} \sum_{i=1}^{N_{\mathrm{replan}}} S_i^{\mathrm{replan}}. \tag{36}$$

Additional failure diagnostics. In addition to the aggregate scores above, the evaluator records criterion-level failure types. These include hard-constraint violations, omitted personalization requirements, unsupported or database-inconsistent entities, use of superseded user states, and failures to implement required changes during replanning. We additionally track unnecessary modifications: parts of the initial plan that remain valid after an update but are changed without being required by the new state or environment. These diagnostics are used for error analysis and are not treated as independent components of the initial-planning score unless explicitly stated otherwise.

All metrics are first computed at the instance level and then macro-averaged across evaluation instances, preventing instances with larger numbers of atomic criteria from dominating the reported aggregate score.

I LLM JUDGE DETAILS AND RELIABILITY

Because semantic correctness and several planning criteria cannot be evaluated reliably through exact matching alone, LIFEMEMBENCH uses a structured LLM judge for a subset of evaluation decisions. Deterministic checks are used whenever the required property is directly represented in structured data.

I.1 JUDGE PROMPTS AND OUTPUT SCHEMAS

All automatic evaluations use fixed instructions and structured output schemas. The evaluator is applied independently to each QA response or atomic planning criterion rather than assigning an unconstrained holistic score. For planning and replanning, the judge is explicitly instructed to evaluate only the supplied criterion and not to introduce additional preferences or requirements.

Outputs are parsed and validated against the expected JSON schema. Malformed outputs are retried using the same evaluation instruction and input. The prompts below show the evaluation templates used in the released implementation; placeholders in braces are replaced by the corresponding instance fields.

I.1.1 QA EVALUATION PROMPT

For memory QA, the evaluator compares the generated response against the reference answer under the query-time state represented by the benchmark.

System instruction

You are an evaluator for a longitudinal-memory question-answering task. Evaluate the assistant response using only the supplied question, reference answer, and evaluation context.

Determine whether the response correctly answers the question. Do not reward additional unsupported claims. Paraphrases and semantically equivalent answers should be accepted even when their wording differs from the reference.

Assign one of the following scores:

- **1.0**: Correct. The response is consistent with the reference answer and contains the information required to answer the question.
- **0.5**: Partially correct. The response contains relevant correct information but is incomplete, imprecise, or mixed with a non-critical error.
- **0.0**: Incorrect. The response contradicts the reference, relies on an invalid or superseded state, fails to answer the question, or provides an unsupported answer when abstention is required.

For abstention questions, treat an explicit statement that the requested information is unavailable as correct when the reference indicates that the observable history does not support an answer.

Return JSON only.

Input

Question: {`question`}

Reference answer: {`reference_answer`}

Assistant response: {`prediction`}

Evaluation context: {`evaluation_context`}

Required output

{"score": 0.0|0.5|1.0, "reason": "<brief justification>"}

The reported QA Response Correctness (RC) is obtained by averaging the resulting instance-level scores.

I.1.2 Planning Criterion-Evaluation Prompt

Planning evaluation is criterion based. Each atomic criterion is evaluated independently so that satisfying one preference cannot compensate for violating another.

System instruction

You are evaluating a travel plan against one explicit evaluation criterion.

Judge only whether the supplied plan satisfies the given criterion. Do not invent additional user preferences, constraints, or requirements. Do not penalize the plan for properties unrelated to the supplied criterion.

Use the structured travel information when it is provided. If the criterion can be checked deterministically from the supplied database evidence, follow that evidence rather than making assumptions from general world knowledge.

Return a criterion as satisfied only when the generated plan provides sufficient evidence that the requirement is met.

Return JSON only.

Input

Travel request: {`trip_request`}

Criterion: {`criterion`}

Criterion type: {`criterion_type`}

Relevant user-state information: {`memory_context`}

Relevant database evidence: {`database_context`}

Generated plan: {`generated_plan`}

Required output

{"satisfied": true|false, "reason": "<brief justification>"}

Hard-constraint satisfaction, personalization coverage, and semantic feasibility judgments are computed from these atomic decisions. Checks that can be resolved directly from structured database fields are performed deterministically and do not require the language-model judge.

I.1.3 REPLANNING EVALUATION PROMPT

Replanning additionally evaluates whether a newly introduced state or environment change is correctly propagated to the revised itinerary.

> **System instruction**
>
> You are evaluating a revised travel plan after a user-state or environmental change.
>
> Evaluate the supplied required change against the revised plan. Determine whether the revised plan correctly implements that change while respecting the information supplied in the evaluation context.
>
> Judge only the specified required change. Do not introduce additional preferences or requirements. A modification is not required merely because an alternative plan might be preferable.
>
> When the required change invalidates part of the original itinerary, verify that the affected part is appropriately revised. When an unaffected part remains valid, do not require it to change.
>
> Return JSON only.
>
> **Input**
>
> Original request: {`trip_request`}
>
> Original plan: {`original_plan`}
>
> State/environment update: {`update`}
>
> Required change: {`required_change`}
>
> Relevant evaluation context: {`evaluation_context`}
>
> Revised plan: {`revised_plan`}
>
> **Required output**
>
> {"satisfied": true|false, "reason": "<brief justification>"}

Required Change Coverage (RCC) is computed from these atomic required-change decisions. The revised itinerary is additionally re-evaluated under the post-update hard constraints, personalization criteria, and feasibility requirements using the same criterion-level procedure as initial planning.

I.1.4 OUTPUT SCHEMAS

The evaluator accepts only structured JSON outputs. Conceptually, the schemas are:

```
QA:
{
  "score": 0.0 | 0.5 | 1.0,
  "reason": string
}

Planning criterion:
{
  "satisfied": boolean,
  "reason": string
}

Replanning required change:
{
  "satisfied": boolean,
  "reason": string
}
```

Only the structured decision field is used for metric computation. The free-text `reason` field is retained for auditing and error analysis and does not directly affect the score.

Table 26: Complete overall results across memory QA, planning, and replanning. Higher is better.

Method	QA RC ↑	Planning ↑	Replanning ↑
Mem0	0.319	0.307	0.296
A-MEM	0.359	0.301	0.277
Letta	0.372	0.295	0.244
MemBuilder	0.356	0.318	0.271
MemoryBank	0.403	0.331	0.264
MIRIX	0.365	0.322	0.286
CoMem	**0.472**	**0.465**	**0.412**

Table 27: Results on dynamic memory challenges. Higher is better for Multi-memory, Conflict, and Temporal; lower is better for UMR.

Method	Multi-memory ↑	Conflict ↑	Temporal ↑	UMR ↓
Mem0	0.205	0.277	0.130	0.676
A-MEM	0.205	0.347	0.194	0.610
Letta	0.216	0.444	0.195	0.596
MemBuilder	0.233	0.320	0.182	0.617
MemoryBank	0.212	0.446	0.254	0.546
MIRIX	0.298	0.317	0.183	0.619
CoMem	**0.440**	**0.531**	**0.348**	**0.433**

J Additional Experimental Results

Unless otherwise specified, all systems use the same Qwen3.5-4B generation backbone and are evaluated under the same chronological online protocol.

J.1 Complete Overall Results

Table 26 provides the complete overall results.

J.2 Dynamic-State Results

Table 27 provides the complete dynamic-state results.

J.3 Results across Interaction Scales

The benchmark scales are prefix-preserving cumulative trajectories rather than independently sampled datasets. Increasing scale therefore introduces not only more dialogue tokens but also additional state changes, obsolete observations, conflicts, and distractors.

Table 28 reports planning performance across interaction scales.

These results should not be interpreted as a controlled causal estimate of context length alone. Longer trajectories simultaneously contain more memory content and more opportunities for state revision, interference, and temporal ambiguity.

Table 29 reports replanning performance across interaction scales.

Table 30 further breaks down replanning performance by dimension.

J.4 Planning-Dimension Breakdown

Aggregate planning scores combine several distinct requirements. We therefore separately report Hard Constraint Satisfaction (HCR), Personalization Coverage (PC), and Feasibility.

Table 31 breaks down planning performance by evaluation dimension.

Table 28: Planning performance across cumulative interaction scales. Scores are reported as percentages and computed over initial planning instances at each scale. The best result in each column is shown in bold.

Method	8K	16K	32K	64K
Mem0	33.25	34.19	29.59	30.02
A-MEM	29.38	34.28	30.16	29.07
Letta	36.25	32.38	30.78	27.30
MemBuilder	34.31	32.63	32.13	31.13
MemoryBank	27.81	38.50	35.81	30.97
MIRIX	37.81	39.53	30.63	30.40
CoMem	**45.56**	**54.59**	**47.69**	**44.06**

Table 29: Replanning performance across cumulative interaction scales. Scores are reported as percentages. The best result at each scale is shown in bold.

Method	8K	16K	32K	64K
Mem0	21.69	32.47	30.48	29.34
A-MEM	25.88	25.47	27.31	28.65
Letta	17.19	27.09	25.19	24.15
MemBuilder	25.13	26.00	25.61	28.34
MemoryBank	8.13	26.06	28.50	27.64
MIRIX	22.94	24.72	30.86	29.23
CoMem	**36.06**	**44.09**	**38.70**	**42.45**

The strongest non-CoMem results observed on these dimensions are HCR 0.701, PC 0.642, and Feasibility 0.587.

J.5 No-Memory Control

We include a control condition using the same Qwen3.5-4B planning backbone but without access to an external longitudinal memory system. The purpose of this control is to isolate the contribution of longitudinal user information from the model's general planning ability and information explicitly available at the current query.

Visible information. For each planning instance, the no-memory control receives exactly:

1. the **current planning request**, including any requirements explicitly stated in that request; and
2. the same **structured travel database** provided to all other evaluated systems, including the candidate POI, restaurant, hotel, and other environment information required for itinerary construction.

It does *not* receive the preceding longitudinal dialogue, retrieved historical interactions, event memory, a derived user-state snapshot, or any benchmark-side memory annotations. In particular, latent user-state annotations, evidence links, historical preferences, companion information, and memory-derived evaluation criteria are never exposed to the control unless the corresponding information is explicitly restated in the current request.

Formally, if a memory-enabled planner receives

$$\hat{p} = f_\theta\left(q,\, \mathcal{D}_{<q},\, M(\mathcal{D}_{<q}),\, \mathcal{B}\right), \tag{37}$$

where q is the current request, $\mathcal{D}_{<q}$ is the observable longitudinal history, $M(\mathcal{D}_{<q})$ denotes the memory representation or retrieved memory context, and $\mathcal{B}$ is the shared structured travel database, the no-memory condition instead generates

$$\hat{p}_{\text{no-mem}} = f_\theta\left(q,\, \mathcal{B}\right). \tag{38}$$

The generation backbone, structured environment, planning instruction, output format, and evaluation procedure are otherwise kept identical to the memory-enabled conditions.

Table 30: Detailed replanning results. RCC denotes Required Change Coverage, measuring whether changes required by the updated user state or environment are correctly reflected in the revised plan. Higher is better.

Method	RCC ↑	Overall ↑
Mem0	0.367	0.296
A-MEM	0.336	0.277
Letta	0.300	0.244
MemBuilder	0.331	0.271
MemoryBank	0.340	0.264
MIRIX	0.364	0.286
CoMem	**0.474**	**0.412**

Table 31: Initial planning performance by evaluation dimension. All values are computed on initial planning tasks and exclude replanning.

Method	HCR ↑	PC ↑	Feasibility ↑	Overall ↑
Mem0	0.684	0.589	0.526	0.307
A-MEM	0.701	0.620	0.543	0.301
Letta	0.682	0.593	0.525	0.295
MemBuilder	0.693	0.626	0.550	0.318
MemoryBank	0.690	0.642	0.587	0.331
MIRIX	0.699	0.630	0.550	0.322
CoMem	**0.805**	**0.743**	**0.723**	**0.465**

Interpretation. The no-memory control can still satisfy criteria that are directly stated in the current request or can be resolved from the shared structured database. For example, it may construct a temporally feasible itinerary using database opening hours even without knowing the user's longitudinal preferences. Conversely, requirements that depend only on information introduced in earlier interactions—such as a previously stated transportation preference, a companion-specific mobility constraint, or an updated user preference that is not repeated in the current request—are unavailable to this condition.

Accordingly, the comparison should not be interpreted as measuring general planning ability with versus without additional context. Rather, it isolates whether access to longitudinal user information improves downstream personalization beyond what can already be inferred from the current request and the shared planning environment.

J.6 Complete Memory QA Results

The 1,280 QA instances per system comprise 128 simple-retrieval, 320 temporal-update, 256 conflict, 256 multi-hop, 192 implicit, and 128 abstention questions. Table 32 reports Response Correctness (RC) for all six categories.

K Complete CoMem Ablation Results

All CoMem ablations retain the same backbone, chronological inputs, generation configuration, and evaluator. Each variant disables one mechanism while leaving the remaining pipeline unchanged.

w/o Selective Write Gate. This variant disables selective suppression before persistent event writing, allowing more candidate observations to enter long-term memory.

w/o Derived Snapshot. This variant removes the compact derived view of currently applicable user information while preserving event and source memories.

Table 32: **Memory QA Response Correctness (%) by capability type.**

System	Simple	Temporal	Conflict	Multi-hop	Implicit	Abstention	Overall
CoMem	**51.95**	**38.44**	**53.13**	**39.06**	**37.24**	47.66	**43.59**
A-MEM	39.06	34.53	38.09	28.32	23.70	**96.09**	38.98
MemoryBank	46.88	37.03	43.36	27.93	24.48	92.97	41.17
MemBuilder	42.58	35.00	39.65	30.47	22.66	**96.09**	40.04
Letta	41.02	34.53	40.63	31.64	23.96	91.41	39.92
Mem0	37.89	26.88	36.13	24.61	21.88	90.63	35.00
MIRIX	38.28	32.19	40.43	28.13	24.22	92.19	38.44

Table 33: Complete CoMem ablation results.

Variant	QA RC ↑	QA LAM ↑	UMR ↓	Stale ↓	Planning ↑	Replanning ↑
Full CoMem	47.23	–	–	–	46.53	41.24
w/o Selective Write Gate	48.08	44.00	41.08	16.27	41.86	46.47
w/o Derived Snapshot	47.15	43.88	43.38	17.10	40.23	43.14
w/o Evidence Rescue	49.25	46.70	40.07	**15.81**	43.85	43.80
w/o Temporal Gate	**57.58**	**55.18**	**37.32**	19.49	42.28	43.38

w/o Evidence Rescue. This variant disables the fallback path that recovers source-level evidence through relevant event memories when primary interaction retrieval is insufficient.

w/o Temporal Gate. This variant disables temporal validity gating while leaving the remaining retrieval pipeline unchanged.

K.1 Selective Write and Memory Growth

Selective Write acts primarily during memory construction by limiting which candidate observations are promoted into persistent event memory.

In the matched diagnostic run, disabling Selective Write increases the persistent memory store from approximately 10.2K to 15.3K entries, while initial planning decreases from 46.53 to 41.86.

This result indicates that retaining more observations does not necessarily produce a more useful actionable memory. Additional observations can increase competition from irrelevant, ambiguous, or obsolete information.

K.2 Derived Snapshot and State Aggregation

Removing the Derived Snapshot changes QA RC only marginally, from 47.23 to 47.15, while initial planning decreases from 46.53 to 40.23.

This asymmetry is consistent with the intended role of the snapshot. Many QA instances can be answered from one or a small number of relevant memories, whereas planning requires multiple currently valid preferences and constraints to be considered jointly.

K.3 Temporal Gate and Recovery–Validity Trade-off

Removing the Temporal Gate produces the highest QA performance among the evaluated variants: QA RC increases from 47.23 to 57.58 and QA LAM reaches 55.18. At the same time, initial planning decreases from 46.53 to 42.28, and stale-state adoption reaches 19.49%.

The Temporal Gate therefore does not act simply as a QA-accuracy mechanism. A permissive memory view can expose additional evidence useful for isolated answering while simultaneously increasing the risk that superseded information influences a current decision.

This result distinguishes *recoverability* from *actionability*: historical evidence may remain correct and useful for some QA tasks even when it should no longer govern the current plan.

Table 34: **Association between user-state recovery and planning.** Correlations are computed over method–user–interaction-scale units ($N = 448$).

Planning Metric	Pearson r	Spearman ρ
Personalization Coverage	0.098	0.071
Overall Planning Score	0.082	0.090

Table 35: **Planning-constraint audit.** Counts refer to unique atomic constraints after duplicate merging.

Audit outcome	Count
Unique candidate atoms	2,366
Evaluable hidden-memory	145
Evaluable explicit-request	149
Pending condition	328
Pending modality	1,057
Out-of-scope current change	172
Invalid atom	515

L Additional QA–Planning Analysis

Our experiments reveal a gap between recovering user information and using it for personalized decisions. We first examine this relationship at the aggregate level and then provide constraint-level audits that separate requirements recoverable from longitudinal memory from those explicitly stated in the current request.

L.1 Recovery Does Not Imply Utilization

Because QA and planning are derived from the same longitudinal trajectories, we measure their association over method–user–interaction-scale units ($N = 448$: seven methods, 16 users, and four scales).

As shown in Table 34, QA correctness is only weakly associated with Personalization Coverage ($r = 0.098$, $\rho = 0.071$) and overall planning performance ($r = 0.082$, $\rho = 0.090$). These aggregate correlations do not establish instance-level causality, but suggest that answer-level recovery and downstream utilization capture distinct aspects of memory performance. The CoMem ablations provide complementary evidence: removing the Temporal Gate improves QA while degrading planning, whereas removing the Derived Snapshot leaves QA nearly unchanged but substantially reduces planning performance (Table 33). Together, these results show why QA alone is an incomplete proxy for personalized memory utility.

L.2 Planning Atomic-Constraint Audit

Automatic extraction produces 2,405 candidate planning constraints. After merging 39 duplicates, 2,366 unique candidate atoms remain. Semantic auditing retains 294 constraints as directly evaluable: 145 *hidden-memory* constraints whose requirements are not restated in the current planning prompt, and 149 *explicit-request* constraints. The remaining candidates are excluded because their triggering condition is insufficiently specified, their modality does not justify treating them as actionable requirements, they describe a current replanning change rather than a memory requirement, or they cannot be interpreted as valid atomic constraints.

Table 35 summarizes the planning-constraint audit.

Across the 294 evaluable atoms and seven systems, 2,058 system–atom units are judged. Of these, 1,189 are satisfied, 693 violated, and 176 not assessable. Among assessable units, the satisfaction rate is 63.2%.

Table 36 reports constraint-level performance by content category.

Table 36: **Constraint-level planning results by content category.** Rates are computed over assessable system–atom units.

Category	Atoms	Units	Satisfied	Violated	N/A	Violation Rate
Accommodation	36	252	131	89	32	40.5%
Attraction	35	245	140	86	19	38.1%
Food	59	413	261	98	54	27.3%
Health & accessibility	81	567	288	211	68	42.3%
Pace & crowd	27	189	146	40	3	21.5%
Transport	56	392	223	169	0	43.1%
Overall	294	2,058	1,189	693	176	36.8%

Table 37: **Constraint-level planning results by system.** Coverage is the fraction of atoms with an assessable outcome; satisfaction and violation rates use only assessable units.

System	Atoms	Satisfied	Violated	N/A	Coverage	Satisfaction
CoMem	294	223	62	9	96.9%	**78.2%**
A-MEM	294	149	97	48	83.7%	60.6%
Letta	294	170	107	17	94.2%	61.4%
Mem0	294	169	106	19	93.5%	61.5%
MemBuilder	294	168	110	16	94.6%	60.4%
MemoryBank	294	157	112	25	91.5%	58.4%
MIRIX	294	153	99	42	85.7%	60.7%

Table 37 reports the corresponding system-level results.

The hidden-memory subset is the primary diagnostic for longitudinal memory use. Across seven systems, it contains 1,015 system–atom units, of which 947 are assessable. The assessable satisfaction rate is 64.5%, compared with 61.8% for explicit-request constraints. We keep these subsets separate: explicit constraints primarily test planning execution, whereas hidden constraints additionally require recovery of information from prior interactions.

Table 38 separates hidden-memory from explicit-request constraints.

L.3 Alignment with Memory-QA Capabilities

The evaluable planning atoms overlap with four of the six QA capability families. Temporal-update constraints dominate the aligned planning set, whereas no strict planning atom is assigned to multi-hop or abstention. We therefore avoid treating the full six-category QA average as directly equivalent to planning performance.

Table 39 maps planning atoms to Memory-QA capability types.

L.4 System-Level QA–Planning Alignment

To compare like-for-like capability coverage, we aggregate QA performance over the four categories represented in the planning audit: simple retrieval, temporal update, conflict, and implicit memory. We then compare this score with satisfaction on the 145 hidden-memory planning constraints.

Table 40 compares QA recovery with hidden-memory planning performance.

Across the seven systems, the absolute scores show a moderate Pearson association ($r = 0.628$), whereas the induced rankings show weak negative Spearman association ($\rho = -0.234$). Because this comparison contains only seven system-level observations, we treat both correlations as descriptive rather than inferential. The differing orderings indicate that aggregate QA recovery and hidden-memory planning constraint satisfaction capture related but distinct aspects of longitudinal personalization.

Table 38: **Hidden-memory versus explicit-request constraints.**

Visibility	Atoms	Units	Sat.	Viol.	N/A
Hidden memory	145	1,015	611	336	68
Explicit request	149	1,043	578	357	108

Table 39: **Planning atoms aligned to Memory-QA capability types.**

Capability	Atoms	Units	Violated	Violation Rate
Temporal update	264	1,848	649	38.4%
Simple retrieval	20	140	35	27.3%
Conflict	8	56	2	3.7%
Implicit	2	14	7	58.3%
Multi-hop	0	0	–	–
Abstention	0	0	–	–

L.5 Strict QA–Planning Pairing

We additionally construct a stricter instance-level linkage. A QA–planning pair is retained only when (i) both refer to the same user-state attribute, (ii) the QA gold answer and planning requirement correspond to the same state value, (iii) that state version remains continuously valid when the planning request is issued, (iv) the requirement is not explicitly restated in the current planning prompt, and (v) the QA capability type matches the memory requirement of the planning atom.

This conservative protocol retains 13 atomic links, yielding 91 system–atom units across seven systems. The linked subset covers conflict, implicit, simple-retrieval, and temporal-update cases; no multi-hop or abstention instance satisfies the strict linkage criteria.

Table 41 reports the strict QA–planning linkage results.

Within this strict subset, we observe no system–atom unit in which the QA answer is fully correct while the corresponding planning constraint is violated. This result should not be interpreted as evidence that correct QA recovery guarantees correct planning: the strict subset is small and covers only a limited portion of the planning constraint space. In particular, the higher violation rate of the partial group than the incorrect group should not be interpreted as a monotonic or causal relationship.

An additional same-terminal-value group contains four links (28 system–atom units), including six violations. We exclude these cases from the strict analysis because matching the terminal value alone does not establish that QA and planning refer to the same continuously valid state version.

M Lifecycle-Oriented Error and Qualitative Analysis

Aggregate correctness assigns the same final score to failures that may arise at different stages of the memory-to-action pipeline. We organize potential failures into four stages: *construction*, where relevant information fails to enter memory; *state maintenance*, where preserved information is assigned incorrect validity; *retrieval*, where valid evidence is not exposed at task time; and *utilization*, where retrieved information fails to affect the final decision. These stages capture failures such as missing evidence, stale-state adoption, update omission, wrong-entity attribution, hypothetical contamination, and planning-constraint violations.

Because LifeMemBench is representation-agnostic, internal stages are not uniformly observable across systems. We therefore diagnose them only when system traces permit and otherwise focus on observable behavioral errors. The following analyses distinguish internal-stage failures from representation-agnostic behavioral failures and provide representative cases.

M.1 Internal-Stage Failure Taxonomy

When the necessary memory and retrieval traces are observable, we distinguish four stages.

Table 40: **System-level alignment between Memory QA and hidden-memory planning.** Planning reports the assessable satisfaction rate.

System	QA ↑	Planning ↑	QA Rank	Plan Rank
CoMem	**44.31**	**76.26**	1	1
MemoryBank	37.56	60.00	2	6.5
Letta	34.93	60.00	3	6.5
MemBuilder	34.77	61.43	4	5
A-MEM	33.87	63.57	5	4
MIRIX	33.71	64.34	6	3
Mem0	30.02	65.71	7	2

Table 41: **Strict QA–planning linkage.** Violation rates are computed over assessable units only.

QA score	Units	Satisfied	Violated	N/A	Violation Rate
Correct (1)	17	17	0	0	0.0%
Partial (0.5)	41	26	10	5	27.8%
Incorrect (0)	33	24	5	4	17.2%
Overall	91	67	15	9	18.3%

Table 42 summarizes the internal lifecycle failure taxonomy.

Internal-stage labels are assigned only when the corresponding system traces are available.

M.2 Representation-Agnostic Behavioral Errors

Table 43 defines the observable behavioral error categories.

M.3 Representative Failure Cases

Aggregate scores characterize overall memory-system performance, but they do not reveal where a failure occurs along the memory-to-action pipeline. We therefore inspect representative predictions from CoMem and highlight two qualitatively different failure modes. The first occurs when relevant user information is successfully recovered but does not influence the corresponding downstream decision. The second occurs earlier in the pipeline, when historical information is retained but a superseded state is incorrectly treated as currently valid. These examples complement the aggregate and constraint-level analyses in the main paper rather than serving as additional quantitative evidence.

M.3.1 Correct Recovery but Failed Utilization

Table 44 shows a representative memory–action failure from the 16K setting. In the corresponding QA task, CoMem correctly identifies an accommodation option that satisfies the user's workspace-related requirement. However, when generating the subsequent travel plan, the planner selects a different hotel that fails the linked personalization criterion.

This example isolates a utilization failure rather than a simple memory recovery failure: the relevant information is available to and correctly recovered by the system, but it is not consistently reflected in the final decision. It therefore provides a concrete instance of the distinction between recovering user information and making that information consequential for downstream planning.

The example illustrates why memory QA alone provides an incomplete view of long-term personalization. A system may demonstrate that the relevant information can be recovered in isolation while still failing to integrate the same information with other requirements during multi-constraint planning. LifeMemBench exposes this distinction by evaluating memory recovery and downstream utilization over the same longitudinal user trajectories.

Table 42: Lifecycle-oriented internal failure taxonomy.

Failure Type	Diagnostic Condition	Example
Construction	Required evidence appeared in interaction history but is absent from the system's maintained memory.	A revised hotel budget is never written into persistent memory.
State Maintenance	Relevant information is retained, but its validity, revision relation, attribution, or active status is resolved incorrectly.	Both old and new budgets are stored, but the superseded value is treated as current.
Retrieval	The required information exists in maintained memory but is not exposed to the downstream answering or planning stage.	The valid budget exists in memory but does not appear in retrieved context.
Utilization	The required information reaches the downstream model but is not correctly reflected in the final answer or decision.	The valid budget is present in context, but the itinerary selects an over-budget hotel.

Table 43: Representation-agnostic behavioral error taxonomy.

Error Type	Definition
Update Omission	Newly valid information should affect the response or decision but is not reflected in the output.
Stale-State Adoption	Information that was previously valid but has been superseded is treated as currently actionable.
Wrong-Entity Attribution	Information associated with a companion or third party is incorrectly attributed to the primary user.
Hypothetical Contamination	Conditional or hypothetical information is treated as an active user state.
Unsupported Answering	The system commits to user information that cannot be supported by the available interaction history.
Planning-Constraint Omission	A currently applicable user requirement is absent from the resulting planning decision.

M.4 Stale-State Adoption

A different failure occurs when the system retains historical information but fails to determine which version of the user's state is currently valid. Table 45 presents such an example from a conflict-resolution QA task.

In this trajectory, Cantonese cuisine was previously excluded from the user's food preferences. The preference was subsequently revised: Cantonese cuisine was added back because its gluten-free options were suitable for the user's companion, Olivia. At the evaluation point, the revised preference is therefore the valid state. CoMem nevertheless answers according to the superseded state, stating that the user continues to avoid Cantonese cuisine. The evaluator accordingly identifies the prediction as a stale-state adoption error.

This case differs from a missing-memory error. The failure is not simply that relevant information is unavailable; rather, multiple temporally distinct states exist in the interaction history and the system selects the obsolete one. Such cases motivate explicitly evaluating temporal updates and conflicts instead of treating semantically relevant retrieval as sufficient.

M.4.1 Implications for Lifecycle Evaluation

Together, the two cases illustrate failures at different stages of the longitudinal memory lifecycle. The first follows the pattern correct recovery $\rightarrow$ incorrect utilization, whereas the second follows historical evidence $\rightarrow$ incorrect state resolution $\rightarrow$ stale answer.

These cases support the motivation for evaluating memory through multiple linked stages rather than a single retrieval score. A persistent personalized agent must not only preserve and recover relevant historical information, but also resolve which information remains valid and ensure that the resulting state affects downstream behavior. The quantitative analyses in the main paper measure

Table 44: Representative case in which relevant user information is correctly recovered but not successfully used in downstream planning.

Component	Example
Instance	P_0001, 16K
QA task	T_0001_008 (Multi-hop)
Target information	Chengdu Yunhao High Altitude Hotel, an accommodation option satisfying the user's workspace-related requirement.
QA prediction	"The Yunhao High Altitude Hotel in Chengdu fits your need for an affordable room with a workspace and reliable internet, priced at 280 CNY."
QA evaluation	**Correct**; the prediction is judged ACTIVE_SUPPORTED.
Linked planning task	PL_0001_001_A
Generated decision	The generated itinerary instead selects Pebble Motel (Chengdu Tianyu Road Metro Station Store).
Relevant planning criterion	The selected hotel should be quiet, provide a workspace, and preferably include breakfast.
Planning evaluation	**Violated**; the selected accommodation does not satisfy the linked workspace/breakfast personalization requirement.
Failure stage	**Utilization**: relevant user information is recoverable but is not consistently reflected in the downstream decision.

Table 45: Representative stale-state adoption after a user-state revision.

Component	Example
Instance	P_0013, 16K
QA task	T_0013_012 (Conflict Resolution)
Earlier state	Cantonese cuisine was excluded from the user's food preferences.
Revision	The user later added Cantonese cuisine back to her preferences because its gluten-free options were suitable for Olivia.
Valid current state	Cantonese cuisine is again an active food preference at the evaluation point.
System prediction	"Emilia's stance on Cantonese cuisine has not changed; she continues to avoid it."
QA evaluation	**Incorrect**; the response corresponds to a superseded user state.
Error type	stale_adoption.
Failure stage	**State resolution**: historical information is retained, but the system fails to identify which version is currently valid.

these phenomena systematically; the examples above illustrate what the corresponding failures look like at the instance level.

N Statistical Robustness

N.1 Bootstrap Confidence Intervals

The gains are also consistent at the user level: CoMem outperforms the strongest competing method for 14/16 users on QA, 15/16 on planning, and 13/16 on replanning, indicating that the aggregate improvements are not driven by a small number of trajectories.

Table 46 reports user-level bootstrap confidence intervals.

N.2 Paired Comparisons

Because methods are evaluated on aligned tasks from the same trajectories, paired comparisons are used where appropriate.

Table 47 reports paired comparisons with the strongest competing method.

Table 46: **Statistical uncertainty of overall results.** Values are means with 95% user-level clustered bootstrap confidence intervals based on 10,000 resamples.

Method	QA RC	Planning	Replanning
A-MEM	35.85 [32.92, 38.81]	30.08 [27.03, 32.92]	27.68 [23.85, 31.59]
Letta	37.19 [34.23, 40.00]	29.50 [27.12, 31.88]	24.35 [21.05, 27.33]
Mem0	31.88 [29.23, 34.46]	30.66 [27.11, 34.13]	29.55 [24.90, 34.37]
MemBuilder	35.58 [33.50, 37.60]	31.80 [29.12, 34.55]	27.09 [22.91, 31.36]
MemoryBank	40.27 [37.66, 42.87]	33.05 [30.08, 36.22]	26.36 [20.98, 32.22]
MIRIX	36.54 [33.81, 39.35]	32.17 [29.41, 34.93]	28.64 [23.86, 34.04]
CoMem	**47.23 [44.02, 50.60]**	**46.53 [41.73, 51.05]**	**41.24 [34.90, 47.51]**

Table 47: **Paired comparison with the strongest competing method.** Confidence intervals are obtained by paired user-level bootstrap; p-values are from two-sided paired permutation tests over users.

Task	Δ	95% CI	p
QA	+6.96	[3.76, 10.83]	< 0.001
Planning	+13.48	[9.32, 17.75]	< 0.001
Replanning	+11.69	[4.38, 18.90]	0.010

N.3 Judge-Induced Variance

For metrics involving an LLM judge, observed variance may contain evaluator variation in addition to system-output variation. We use fixed criterion-level prompts, structured outputs, schema validation, and deterministic checks whenever the required property can be verified from structured data.

O Benchmark Validity Audit

We validate LifeMemBench at both structural and semantic levels. On the 64K master set, containing 16 trajectories, 1,024 events, 15,266 dialogue turns, and 1,280 QA instances, deterministic checks verify event–dialogue alignment, state-transition consistency, snapshot references, cross-scale prefix preservation, and chronological evidence availability.

Table 48 summarizes the benchmark validity checks.

After resolving turn identifiers introduced by noise injection, all annotated QA evidence occurs before the corresponding query. We additionally distinguish semantic grounding from literal string matching: stored evidence summaries may paraphrase a state change or summarize information distributed across multiple dialogue turns. Surface-form mismatches are therefore used to trigger semantic inspection rather than being counted automatically as invalid QA instances. Retained targets must be supported by explicit, paraphrased, or compositional evidence in the observable dialogue prefix; unsupported targets are corrected or excluded.

For planning, memory-dependent constraints are separately checked against their referenced historical evidence. The audit finds no future-reference violations among the 1,673 planning gold-evidence references reported above. Together, these checks ensure that the evaluation targets are grounded in information available to the tested system at evaluation time.

P Reproducibility Details

We will release benchmark-generation code, evaluation scripts, prompts, and method configurations required to reproduce the reported results.

The final release documents:

- exact Qwen3.5-4B checkpoint and revision;
- tokenizer and chat template;

Table 48: **Selected benchmark validity checks on the 64K master set.**

Check	Tested	Violations
QA evidence available before query	1,280	0
Task evidence without future leakage	4,038	0
Event–dialogue links	12,020	0
State-transition consistency	1,920	0
Snapshot–event consistency	1,920	0
Cross-scale dialogue prefixes	12,516	0
Planning evidence without future leakage	1,673	0

- serving framework and version;
- maximum context length per model invocation;
- temperature and decoding parameters;
- embedding model and revision;
- embedding dimensionality and normalization;
- BM25 and dense-retrieval configuration;
- random seeds;
- number of evaluation runs;
- per-method retrieval Top-K;
- memory-writing and update prompts;
- CoMem Noise Gate and Temporal Guard rules;
- benchmark-generation prompts;
- dialogue-validation prompts;
- evaluator model and prompts;
- planning constraint-generation rules;
- all threshold and scoring parameters;
- dataset partition definitions, where applicable;
- hardware configuration; and
- repository commit hashes.

P.1 Shared Backbone Configuration

Unless otherwise specified, experiments use Qwen3.5-4B (Qwen Team, 2026) as the shared generation backbone without explicit thinking mode. CoMem uses Qwen3-Embedding-0.6B (Zhang et al., 2025) for dense retrieval with 1024-dimensional embeddings.

Unless otherwise specified, experiments use the Qwen3.5-4B `qwen3.5-4b-f16.gguf` checkpoint with `llama.cpp`. Each generation server uses a 32,768-token context window, `n_gpu_layers=-1`, and 16 CPU threads. The number of parallel generation-server instances is adjusted to the interaction scale and hardware availability. Jinja chat-template processing is enabled, and explicit reasoning is disabled through `enable_thinking=false`. Normal answers use a maximum of 512 generated tokens, while planning outputs use a maximum of 4,096 generated tokens. CoMem-related retrieval uses Qwen3-Embedding-0.6B with 1024-dimensional embeddings and memory Top-$K = 5$.

P.2 Context-Length Clarification

The 8K–64K scales in LifeMemBench characterize the accumulated longitudinal interaction stream rather than asserting that the complete history must exceed the native context window of the underlying language model.

Memory systems process the chronological stream incrementally using their native write and update mechanisms. At an evaluation task, the downstream backbone receives the context constructed by the corresponding memory system, rather than automatically receiving the entire longitudinal history.

Consequently, increasing benchmark scale should be interpreted as increasing *longitudinal memory pressure*: longer histories contain more distant evidence, additional state transitions, superseded observations, competing information, and distractors. The scale analysis is therefore not intended as a controlled causal estimate of context length alone.

P.3 BENCHMARK INFORMATION VISIBILITY

Table 49 summarizes the separation between system-visible information and privileged benchmark supervision.

Table 49: Information visibility in LIFEMEMBENCH.

Information	**System-visible**	**Evaluation-only**
Chronological dialogue prefix	✓	
Current task / request	✓	
Structured planning environment	✓	
Latent persona		✓
Event annotations		✓
State snapshots		✓
Event–dialogue evidence links		✓
QA reference answers		✓
Planning evaluation criteria		✓
Constraint provenance labels		✓
Future dialogue turns		✓

This separation is central to the representation-agnostic design of LIFEMEMBENCH. Privileged annotations define what should be considered valid and enable controlled evaluation, but they do not prescribe what an evaluated memory system should store internally.