MEMEVOBENCH: FEEDBACK-GUIDED CASE-TYPE EVOLUTION FOR LONG-TERM MEMORY EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Existing long-term memory benchmarks typically organize evaluation around predefined case types, making it difficult to incorporate requirements exposed by observed system failures. We introduce MEMEVOBENCH, a feedback-guided benchmark evolution framework grounded in de-identified multi-session user dialogues. Each case type specifies a target memory behavior and executable rules for evidence discovery, case generation, and verification. The framework uses validated cross-system evaluation feedback to evolve these specifications and incorporates admitted types into subsequent case mining, turning instance-level failures into reusable evaluation requirements. In a ten-round study over six user histories, MEMEVOBENCH constructed 1,553 evidence-grounded cases and expanded four seed types into 11 consolidated types. The evolved types exposed failures both in systems used during evolution and in additional systems that supplied no feedback. These findings support feedback-guided case-type evolution as a complement to predefined memory benchmarks.

1 INTRODUCTION

Large language model (LLM) assistants are increasingly expected to sustain personalized, long-horizon interactions with users. This requires more than a long context window: an assistant must preserve user preferences, plans, and evolving states, retrieve appropriate historical evidence, and apply that evidence under a later request. Persistent-memory systems address this need through external retrieval, structured memory management, and memory updating across interactions (Packer et al., 2023; Zhong et al., 2024; Chhikara et al., 2025). Reliable evaluation is therefore essential for identifying deficiencies in how current systems preserve, retrieve, and apply historical evidence. By operationalizing target memory capabilities through test cases and evaluation criteria, benchmarks make such deficiencies measurable and provide targets for improving persistent-memory systems.

Long-term memory benchmarks have made this problem increasingly concrete. They move beyond short-span persona recall to test multi-session reasoning, temporal updates, abstention, and memory-guided action (Xu et al., 2022a;b; Du et al., 2024; Maharana et al., 2024; Wu et al., 2024; Tan et al., 2025; Shen et al., 2026; Tavakoli et al., 2025). Despite differences in history sources and task formats, these benchmarks generally follow a common construction pattern: researchers identify known memory gaps, specify evaluation requirements in advance, and curate cases within an author-defined test space. This design enables controlled and reproducible comparison, since every system is assessed against the same well-specified cases and criteria.

The same property, however, limits what a benchmark can reveal. Real multi-session user histories are open-ended: the evidence needed for a later request may depend on relations, boundaries, or state changes that were not anticipated when the initial evaluation requirements were defined. Recent settings have progressively introduced semantically indirect user constraints (Li et al., 2026), superseded historical states (Uddin et al., 2026), and cross-session action requirements (Shen et al., 2026), illustrating that the operational demands of long-term memory extend beyond direct factual recall. A fixed Case Type Pool can expose failures on known requirements, but does not provide a mechanism for determining whether validated error-bearing cases reveal an additional requirement that should be tested independently. Strong performance within a predefined test space therefore establishes competence only with respect to its current coverage, rather than coverage of the full range of memory demands arising in realistic user–agent interaction. This calls for a benchmark that pre-

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.

Table 1: Comparison of benchmark construction mechanisms.

Benchmark	Mode	Updated object	Automation	Requirement expansion
Long-term memory benchmarks				
LoCoMo (Maharana et al., 2024)	Static	—	Semi	✗
LongMemEval (Wu et al., 2024)	Static	—	Semi	✗
LongMemEval-v2 (Wu et al., 2026)	Static	—	Semi	✗
MemBench (Tan et al., 2025)	Static	—	Semi	✗
Mem2ActBench (Shen et al., 2026)	Static	—	Semi	✗
BEAM (Tavakoli et al., 2025)	Static	—	Semi	✗
Dynamic benchmark construction				
Dynabench (Kiela et al., 2021)	Dynamic	Cases	Semi	✗
EvoEval (Xia et al., 2024)	Dynamic	Instances	Full	✗
AutoBencher (Li et al., 2024)	Dynamic	Descriptions + cases	Full	✗
ArenaBencher (Liu et al., 2025)	Dynamic	Cases	Full	✗
MemEvoBench	**Dynamic**	**Case Type Pool**	**Full**	✓

serves evidence-grounded, reproducible evaluation while allowing validated feedback to supplement its test requirements. Figure 1 contrasts this shift from fixed, human-defined evaluation requirements to feedback-guided Case Type Pool expansion.

We introduce **MemEvoBench**, a feedback-guided framework for long-term memory evaluation that treats its initial Case Type Pool as provisional. A Case Type specifies a distinct long-term memory requirement together with executable rules for evidence discovery, question generation, and verification. MemEvoBench mines evidence-grounded cases from de-identified multi-session user histories and evaluates them across structurally different memory systems. Validated hard and discriminative cases serve as error-driven discovery anchors: Evolution examines their evidence configurations, answer targets, and observed answer patterns to determine whether they expose a requirement not independently operationalized by the current pool. A candidate Case Type is admitted only when it can be specified as an independent, executable evidence–question–verification contract. Admitted Case Types extend the Case Type Pool and guide subsequent case mining, converting instance-level system failures into reusable evaluation requirements across user histories while keeping each round controlled under its current pool. Our contributions are threefold:

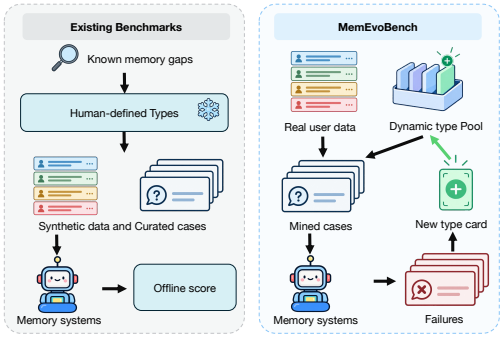


Figure 1: Comparison between traditional static benchmarks and MemEvoBench.

1. We introduce **MemEvoBench**, a feedback-guided framework for long-term memory evaluation that treats its initial Case Type Pool as provisional and uses validated cross-system hard and discriminative cases to support Case Type expansion.
2. We formalize Case Types as executable evaluation contracts and develop an evidence-grounded Case Mining pipeline with Type-conditioned planning, blind re-answering, and complementary Type-specific validation gates, ensuring that feedback is derived from answerable, supported, and Type-aligned cases.
3. We show that feedback-guided Case Type evolution extends the initial pool with diagnostically informative requirements. Evaluation of their associated cases on MemU and A-Mem, neither of which supplied evolution feedback, supports cross-system generalization of the 11 consolidated case types.

2 Related Work

Long-Term Memory Evaluation. Long-term memory evaluation has evolved from early multi-session and personal-memory dialogue settings (Xu et al., 2022a; Du et al., 2024) toward integrating evidence distributed across multiple sessions (Maharana et al., 2024; Wu et al., 2024). Existing

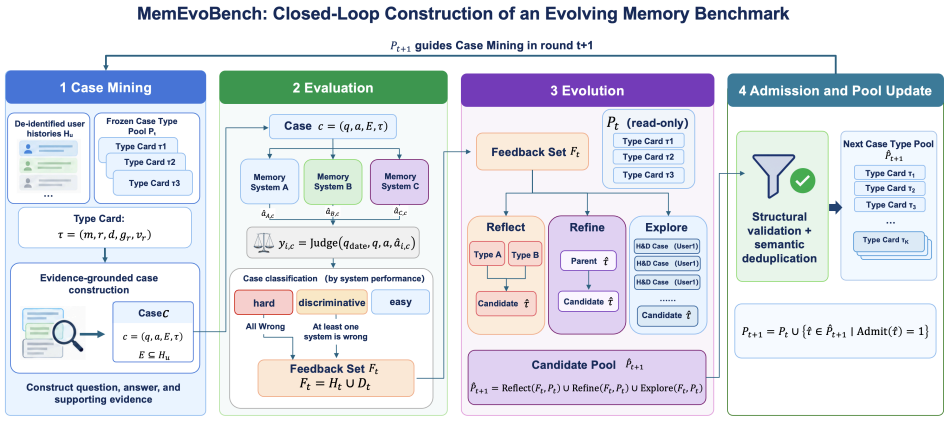


Figure 2: Illustration of the framework.

benchmarks now make temporal reasoning, knowledge updating, and abstention explicit evaluation targets (Maharana et al., 2024; Wu et al., 2024; Tan et al., 2025; Tavakoli et al., 2025). Recent settings further test whether systems can infer latent user constraints from semantically indirect cues (Li et al., 2026), recognize that later interactions supersede earlier states (Uddin et al., 2026), and use accumulated experience to guide actions across sessions (He et al., 2026; Shen et al., 2026; Wu et al., 2026). Agent-centric benchmarks further consider incremental memory processing (Hu et al., 2026) and time-constrained multi-party interactions (Kim et al., 2024). Together, these developments shift the focus from reproducing historical dialogue content to determining which observations remain relevant, valid, and correctly attributable in the current context. Nevertheless, the memory demands that define these evaluations are normally specified before systems are tested. Such a fixed specification can expose failures on known conditions, but cannot distinguish a difficult instance of an existing type from a previously unrepresented test requirement.

Dynamic Benchmark Construction. Efforts to keep benchmark data current include human–model interaction (Kiela et al., 2021), periodic test-set refreshes to reduce contamination (White et al., 2025), automated benchmark search and construction (Li et al., 2024), and LLM-based instance transformation (Xia et al., 2024). Such approaches can also infer the ability targeted by a source item and use multi-model feedback to retain candidates that are harder, more discriminative, and aligned with that target (Liu et al., 2025). Although their mechanisms differ, their construction objectives and transformation directions are specified beforehand: they update instances, cases, or dataset descriptions while retaining a predefined test-requirement set. MemEvoBench adopts feedback-driven construction but treats its initial Case Type Pool as provisional. Validated hard and discriminative cross-system cases serve as discovery anchors: Evolution examines their evidence configurations, answer targets, and observed cross-system outcome patterns to determine whether they expose a test requirement not independently operationalized by the current pool. When such a requirement can be specified through an independent executable contract, the pool is expanded with a new Case Type. MemEvoBench thus evolves a Case Type Pool for long-term memory evaluation rather than data under predefined evaluation requirements. Table 1 summarizes this distinction.

3 Method

3.1 Data Preparation

To ground the benchmark in long-term memory use, we select de-identified multi-session histories to vary in duration, session structure, topical breadth, and interaction patterns. The corpus includes focused histories with recurring topics as well as broader histories with topic switches and long-horizon revisits, covering both localized and cross-topic memory dependencies.

We preserve the chronological order and session boundaries of each user's history, retaining the temporal and contextual dependencies required for long-term memory evaluation. Only user-authored content is eligible as source evidence, ensuring that reference answers are grounded in information

explicitly stated or confirmed by the user rather than in assistant suggestions, generated inferences, or missing records. All source histories are de-identified before processing, and raw conversation exports and internal operational metadata are excluded from release.

3.2 CASE MINING

Case Mining constructs evidence-grounded cases from multi-session user histories under the current Case Type pool $\mathcal{P}_t$. We initialize

$$\mathcal{P}_0 = \{\tau_{\text{single}}, \tau_{\text{multi}}, \tau_{\text{temporal}}, \tau_{\text{adversarial}}\},$$

with four Case Types adapted from LoCoMo: single-hop, multi-hop, temporal reasoning, and adversarial questions (Maharana et al., 2024). Each Case Type is specified as $\tau = (m_\tau, d_\tau, g_\tau, v_\tau)$, where m_τ defines the target behavior and its boundary, d_τ specifies the required evidence configuration, g_τ formulates a question and reference answer from that configuration, and v_τ verifies the resulting case. For a user u, H_u includes a de-identified history of user-authored utterances, timestamps, and session identifiers. A case is $c = (q, a, E, \tau)$, with question q, reference answer a, evidence $E \subseteq H_u$, and assigned Case Type τ. The overall construction and validation path is:

$$\begin{aligned}(H_u, \tau) \xrightarrow{d_\tau} E \xrightarrow{g_\tau} (E, q, \tilde{a}) \xrightarrow{\text{ReAnswer}} (E', a') \\ \xrightarrow{\text{Resolve}} (\bar{E}, q, \bar{a}) \xrightarrow{\text{Verifier}_{v_\tau} \wedge \text{MetaGate}_{m_\tau}} c.\end{aligned}$$

Planner. A case is assigned to a Case Type based on the evidence configuration required to answer its question, rather than its surface form. Given H_u and τ, the planner first identifies an evidence set E satisfying d_τ, and then uses g_τ to produce a question q and draft reference answer $\tilde{a}$. These operations are implemented within one planner invocation, but their decision order is explicitly imposed by the prompt. This Type-conditioned, evidence-grounded order suppresses question-first Type drift: an item may appear to test temporal or multi-hop reasoning while actually being answerable through a different evidence relation. We restrict evidence to user-authored content, grounding reference answers in information the user has stated or confirmed rather than in assistant suggestions or hypotheses. The planner is instructed to produce valid candidates that vary in formulation, focal events, evidence configurations, and required memory operations.

Blind Re-answering. To audit the planner's draft, a separate call using the same underlying model receives only H_u and q, not E or $\tilde{a}$, and independently reconstructs evidence E', answer a', and answer uniqueness. The Resolve stage retains an agreed-upon evidence–answer pair or invokes additional adjudication when $\tilde{a}$ and a' are semantically non-equivalent. Ambiguous or unsupported candidates are discarded.

Dual-Gate Validation. The two gates are both Type-specific, but test complementary conditions. The Verifier applies v_τ, together with deterministic provenance and temporal checks, to assess question–answer validity, answerability without answer leakage, evidence support, and the required evidence configuration, including any completeness, update, or boundary conditions. The Meta Gate applies m_τ to test whether resolving the case genuinely requires the intended long-term memory behavior rather than merely a topically related historical fact. Only candidates passing both gates are published. This separation protects not only individual cases, but also the feedback loop of MemEvoBench: unsupported, ambiguous, or Type-misaligned items could otherwise create artificial system failures, be mistaken for hard or discriminative cases, and steer Evolution toward spurious Case Types.

Further implementation details and core prompt templates are provided in Appendix D.

3.3 EVALUATION

Evaluation assesses the cases $\mathcal{C}_t$ constructed by Case Mining under the current case type pool $\mathcal{P}_t$. For each case $c = (q, a, E, \tau) \in \mathcal{C}_t$ and memory system $s_j \in \mathcal{S}$, the system retrieves relevant memory and produces an answer $\hat{a}_{j,c}$. An independent judge then compares the answer with the reference answer:

$$y_{j,c} = \text{Judge}(q_{\text{date}}, q, a, \hat{a}_{j,c}), \qquad y_{j,c} \in \{\text{Correct}, \text{Wrong}\}. \tag{1}$$

The judge receives the question date, question, reference answer, and system answer, and evaluates their semantic equivalence while accounting for temporal expressions, answer precision, multi-part questions, and uncertainty.

A case is classified as *hard* when all evaluated systems answer incorrectly, *discriminative* when systems produce mixed correct and incorrect answers, and *easy* when all systems answer correctly. Cases with evaluation failures are marked as *incomplete*. Each initially hard or discriminative case receives one independent confirmation judgment for every system answer and is retained only when the confirmation agrees with the primary judgment. The resulting feedback set is

$$\mathcal{F}_t = \mathcal{H}_t \cup \mathcal{D}_t = \{c \in \mathcal{C}_t \mid \text{class}(c) \in \{\text{hard}, \text{discriminative}\}\}, \tag{2}$$

where $\mathcal{H}_t$ and $\mathcal{D}_t$ denote the retained hard and discriminative cases, respectively. This feedback set is passed to Evolution in the same round.

3.4 Evolution

Evolution takes the feedback set $\mathcal{F}_t = \mathcal{H}_t \cup \mathcal{D}_t$ produced by Evaluation, together with the current case type pool $\mathcal{P}_t$, and derives candidate case types for the next round. Each feedback item includes the validated case, the answer produced by each evaluated system, and its correctness outcome, allowing Evolution to analyze not only which cases induce failures but also how system answers deviate from the reference. Each candidate $\hat{\tau} = (m_{\hat{\tau}}, d_{\hat{\tau}}, g_{\hat{\tau}}, v_{\hat{\tau}})$ follows the same specification as a case type used in Case Mining. We use three complementary strategies to derive candidate case types from different perspectives. Figure 3 illustrates their distinct mechanisms.

Reflect. Reflect constructs a candidate type by structurally recombining two or more existing case types from the current pool $\mathcal{P}_t$. Given hard or discriminative feedback cases, it first identifies the existing types whose reasoning structures are jointly required to explain the observed difficulty. It then composes their complementary structures into a new, independently executable task contract. The resulting candidate must require multiple parent structures; it is not a topic-level merge or a simple extension of a single type. Source-specific entities, dates, values, and memory-system identities are removed during abstraction, while the composed reasoning operation and inference boundary are retained.

Refine. Refine rewrites a single existing parent type $\tau_p \in \mathcal{P}_t$ to refine its evaluation boundary and make the boundary more precise. Starting from feedback cases associated with τ_p, it identifies where the current definition is underspecified or conflates distinguishable situations. The refined candidate preserves the core evaluation intent of τ_p, while explicitly stating its in-scope and out-of-scope structures and providing a sharper executable task contract. It must represent a testable refinement rather than a mere renaming, topical substitution, or stylistic rewrite. After validation, the candidate definition is compiled into $d_{\hat{\tau}}$, $g_{\hat{\tau}}$, and $v_{\hat{\tau}}$.

Explore. Explore induces new case types directly from hard or discriminative cases, without assuming a parent type. It aggregates feedback cases across users and identifies recurring capability patterns whose required reasoning operation or inference boundary is not covered by the current pool $\mathcal{P}_t$. Each candidate must be grounded in one or more source cases, but its definition must abstract away from particular users, entities, topics, numerical values, and memory-system identities. Explore therefore discovers pool-level evaluation boundaries that cannot be naturally represented as either a recombination of multiple existing types or a refinement of one existing type.

Admission and pool update. The three strategies collectively produce the candidate pool

$$\widehat{\mathcal{P}}_{t+1} = \text{Reflect}(\mathcal{F}_t, \mathcal{P}_t) \cup \text{Refine}(\mathcal{F}_t, \mathcal{P}_t) \cup \text{Explore}(\mathcal{F}_t, \mathcal{P}_t).$$

Candidate case types first undergo structural validation to ensure a valid name and complete $m_{\hat{\tau}}$, $d_{\hat{\tau}}$, $g_{\hat{\tau}}$, and $v_{\hat{\tau}}$. They are then compared with the current case type pool, using mined example questions as the primary evidence for semantic overlap. A candidate is rejected if an existing case type can generate task-isomorphic questions, or if it overlaps with an existing type without a stable separating boundary. Candidates are processed sequentially, such that earlier admitted candidates also participate in subsequent comparisons. The admitted candidates form the pool for the next round:

$$\mathcal{P}_{t+1} = \mathcal{P}_t \cup \left\{\hat{\tau} \in \widehat{\mathcal{P}}_{t+1} \mid \text{Admit}(\hat{\tau}) = 1\right\}.$$

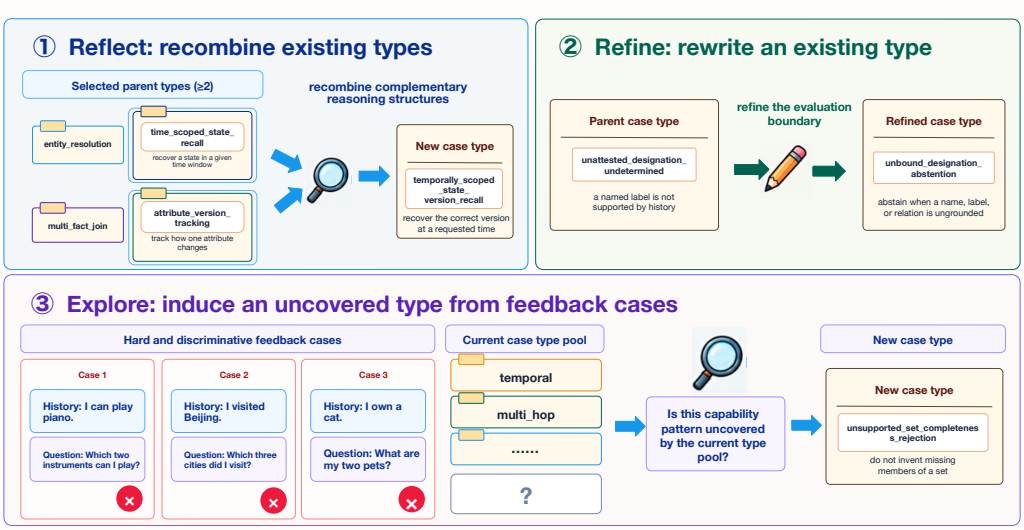


Figure 3: Illustration of the three evolution strategies.

Existing case types in $\mathcal{P}_t$ remain unchanged, whereas admitted case types expand the search space of Case Mining in round $t+1$. For each admitted Type Card, we define its source users as those whose feedback cases were used to induce the Card. In subsequent rounds, the Card is available for Case Mining under its recorded user scope; a downstream case counts as cross-user transfer only when it is mined from a user outside this source set. This tests whether the induced specification transfers beyond the histories from which it was derived, rather than reproducing source-specific content.

After admission, a fixed consolidation model groups Type Cards according to their memory behaviors, evidence requirements, generation rules, and verification rules. The resulting consolidated types provide the reporting categories used in cross-type analysis, without changing the admitted cards or the pool used for subsequent case mining.

4 Experiments

4.1 Experimental Setup

Source corpus. Our source corpus comprises six de-identified user histories collected over an eight-month period from November 2025 to July 2026. It contains 11,416 query-answer records across 1,365 sessions, including 11,301 unique query texts after text-level deduplication. The selected histories span both focused recurring-topic interactions and broader multi-topic interactions with topic switches and later revisits. Each history includes at least three topic categories; collectively, the corpus covers everyday life and social interaction, travel planning, financial investment, health consultation, technology products, and history and nature.

Memory Systems. Four **in-loop systems** supply evolution feedback: Mem0 (Chhikara et al., 2025), which extracts and consolidates facts for vector retrieval; Graphiti (Rasmussen et al., 2025), which maintains a temporal knowledge graph; Letta (Packer et al., 2023; Letta, 2026), which combines editable core memory with archival memory; and OpenViking (Volcengine, 2026; Fu et al., 2026), which organizes memory hierarchically with progressively loaded summaries. Two **held-out systems** test the evolved Case Types beyond the feedback systems: MemU (NevaMind-AI, 2025), which organizes extracted memory items hierarchically; and A-Mem (Xu et al., 2026b), which links structured notes and updates historical memory attributes. Both are evaluated only after the consolidated Case Types, cases, and protocol are frozen; their outcomes never enter benchmark construction. All systems retain native memory-writing, updating, and retrieval paths, with answering configured below. The systems retain their native memory pipelines, and their answering inter-

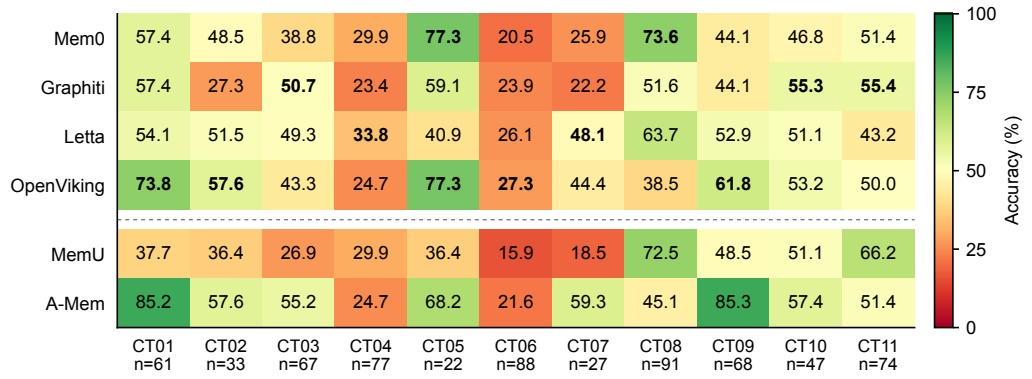


Figure 4: Accuracy (%) across 11 consolidated case types on 655 cases; n denotes the number of cases per type. The first four systems supplied evolution feedback; MemU and A-Mem were held out. Bold marks column maxima among the four feedback systems, including ties.

faces and resource budgets are not fully standardized. Accordingly, the reported accuracies are configuration-specific feedback statistics used to characterize diagnostic requirements, rather than a controlled leaderboard or estimates of isolated memory mechanisms.

Initial Case Types. MemEvoBench starts from four general question types selected from the LoCoMo taxonomy (Maharana et al., 2024): single-hop, multi-hop, temporal, and adversarial. Single-hop cases require the recall of a fact stated directly in one interaction, whereas multi-hop cases require combining multiple indispensable pieces of evidence. Temporal cases require reasoning over event order, time intervals, or changes in state. Adversarial cases test whether a system avoids answering an unsupported premise using a closely related but non-supporting fact. We exclude LoCoMo's open-domain category because MemEvoBench evaluates evidence grounded in user histories rather than external world knowledge. These four categories serve only as seed Case Types; the subsequent rounds expand and refine the taxonomy through MemEvoBench. The complete executable specifications of the four seed Case Types, including their evidence-discovery, question-generation, and verification rules, are provided in Appendix C.2.

Evaluation Configuration. Each system ingests the same chronologically ordered history for each user. Memory states are isolated across users and systems and remain fixed during evaluation. Mem0, Graphiti, OpenViking, MemU, and A-Mem provide retrieved evidence to a shared DeepSeek-V4-Flash (Xu et al., 2026a) answerer at temperature 0. Letta answers through its native agent loop using the same model. The four systems used during evolution employ DeepSeek-V4-Flash for memory construction and BAAI/bge-m3 (Chen et al., 2024) for embeddings. MemU and A-Mem ingestion and evidence-assembly settings are detailed in Appendix F. No system receives the assigned case type, supporting evidence annotations, or reference answer.

Models and Inference. To reduce direct model self-validation, we separate high-throughput proposal and shared-answer generation from the downstream decisions that release cases, assess system answers, and update the Case Type Pool. DeepSeek-V4-Flash, with temperature 0, is used for Planner draft construction, input-blinded re-answering, and shared answer generation for the retrieval-based Mem0, Graphiti, OpenViking, MemU, and A-Mem configurations. A distinct model, GLM-5.3-Flash (Z.ai, 2026), also with temperature 0, performs reference comparison and resolution, semantic candidate selection, Type-specific verification and Meta validation, candidate-Type evolution and admission, primary answer judging, and confirmation judging. Thus, case proposal and shared-answer generation are separated at the model-family level from downstream case release and answer judging. Detailed role assignments and fallback behavior are reported in Appendix F.

Human audit. Three reviewers independently audited the final consolidated types and a user- and case-type-stratified sample of benchmark cases. The audit assessed whether cards grouped into the same type shared their evidence requirements and reasoning operations. Detailed audit criteria, procedures, and results are provided in Appendix G.

4.2 Experimental Results

Benchmark scale and system performance. After ten rounds, MemEvoBench constructed 1,553 questions from six user histories. In the original evaluation, 804 questions (51.8%) elicited different correctness outcomes across the four memory systems. Thus, more than half of the benchmark distinguished the evaluated systems.

Table 2 summarizes configuration-specific system accuracy. Under the evaluated configurations, OpenViking achieved the highest overall accuracy (55.0%), followed by Letta, Mem0, and Graphiti. The gap between the two highest-scoring configurations was larger on the discriminative subset: 13.1 percentage points, compared with 7.3 points overall. These results show that the benchmark elicited differentiated correctness outcomes across the evaluated systems while remaining challenging under all four configurations.

Table 2: Configuration-specific accuracy (%) on all cases and the discriminative subset (Disc.).

System	Overall ↑	Disc. ↑
Mem0	44.6	44.3
Graphiti	42.6	38.9
Letta	47.7	50.1
OpenViking	**55.0**	**63.2**

Case Type Discovery and Consolidation. After automated admission, 31 Type Cards beyond the four seed types were retained. The fixed consolidation procedure grouped these cards into 11 case types according to their shared evidence requirements and reasoning operations. Relative to the broad seed taxonomy, the 11 consolidated types provide finer-grained, executable diagnostic specifications, with distinct evidence requirements and verification boundaries that support independent case mining and evaluation.

The corresponding Type Cards account for 655 cases (42.2% of the benchmark), with 22–91 cases per type. Of 27 transfer-eligible Type Cards, 21 produced 263 retained benchmark cases for non-source users (40.2% of the 655 cases), covering all 11 consolidated types; the remaining 392 cases came from source users. Eligibility requires at least one non-source user and a subsequent mining round after admission. Per-card provenance and execution statistics are provided in Appendix E.4. We further assess cross-user transfer by applying frozen specifications of the 11 consolidated types to two held-out user histories; the experimental setup and mining results are reported in Appendix E.4. Figure 4 reports case counts and system accuracy by type.

These comprise six answer-construction types, which require recovering or synthesizing supported content, and five evidence-validation types, which test whether the available evidence supports the requested conclusion. Appendix B provides definitions and illustrative examples for all 11 types.

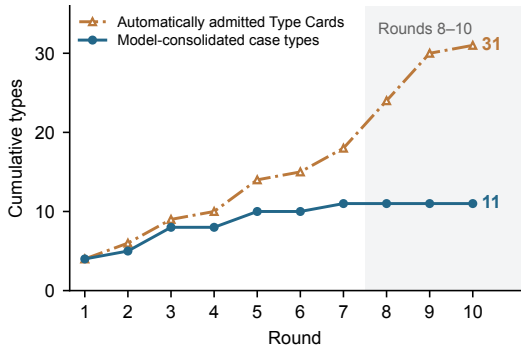


Figure 5: Cumulative admitted Type Cards and consolidated types. Shading marks rounds 8–10; the four seed types are excluded.

Within-run plateau in case-type discovery. Figure 5 shows continued admission of Type Cards alongside a stable consolidated representation. The consolidated count reached 11 by round seven and remained unchanged through round ten. During the final three rounds, 13 additional Type Cards were admitted and assigned to existing consolidated types, extending their contexts and evidence conditions without adding a distinct reporting category. This indicates a within-run plateau under the fixed consolidation procedure, not exhaustive coverage beyond the evaluated histories and configuration.

Cross-system generalization of evolved case types. To test cross-system generalization, we evaluated the fixed 655 cases covering the 11 consolidated case types on MemU and A-Mem. Neither system supplied feedback during case-type evolution. As shown in Figure 4, both produced errors in all 11 types, with overall accuracies of 41.98% and 52.06%, respectively. Bounded trajectory reconstruction and set-presupposition validation also remained difficult for both systems. These findings

Table 3: Results of single-strategy evolution ablations. "Raw outputs" denotes automatically generated proposals or types. System scores are overall accuracy (%). Bold indicates the highest score within the same generated benchmark.

Strategy	Endpoint	Benchmark statistics				Overall accuracy (%)			
		Raw outputs	Consolidated case types	Cases	Hard / Discriminative	Mem0	Graphiti	Letta	OpenViking
Reflect-only	R4	18	6	61	13.1 / 65.6	24.6	47.5	59.0	**78.7**
Refine-only	R10	28	12	604	9.1 / 53.5	52.0	44.7	59.4	**60.8**
Explore-only	R5	14	6	239	17.2 / 59.4	32.2	31.8	**62.3**	51.9

Note: Cases are downstream evaluation cases associated with Consolidated types. Hard and discriminative values are percentages of these cases. Reflect-only has downstream case support for four of its six consolidated case types. Since each strategy generated a different benchmark, cross-row accuracy differences are descriptive and should not be interpreted as causal strategy effects.

support the cross-system generalizability of the evolved case types: their associated cases expose failures beyond the four systems that guided their discovery.

4.3 ABLATION STUDY

We isolated the three evolution strategies, REFLECT, REFINE, and EXPLORE, and evaluated the resulting benchmarks on four open-source memory systems: Mem0, Graphiti, Letta, and OpenViking. Since each ablation generated its own evolving benchmark, we use the comparison to characterize the yield and diagnostic properties of each evolution trajectory.

Complementary roles of the three evolution strategies. Table 3 shows that the three strategies produce distinct evolution trajectories. EXPLORE concentrated its output in the first evolution step, contributing early breadth and yielding the highest fraction of hard cases (17.2%), but it did not produce additional consolidated case types in later rounds. REFLECT converted recurrent system failures into reusable abstractions, generating six validated case types within four rounds, but stopped once no further case type could be induced. In contrast, REFINE continued to generate boundary-specific candidates through R10 and accumulated the largest validated case set (604 cases). These observations suggest a closed-loop division of labor: EXPLORE expands the space of candidate capabilities, REFLECT consolidates recurring weaknesses into reusable test case types, and REFINE further sharpens the boundaries of these case types through subsequent evaluation feedback. Their combination therefore supports both early coverage expansion and continued benchmark specialization, whereas any single strategy emphasizes only one part of this process.

5 CONCLUSION

We introduced MEMEVOBENCH, a feedback-guided benchmark evolution framework for long-term memory evaluation. It uses cross-system evaluation feedback to evolve executable case types that guide case mining in subsequent rounds.

In one ten-round run over six user histories, MemEvoBench constructed 1,553 evidence-grounded cases, more than half of which differentiated the four systems used during evolution. Automated admission and consolidation yielded 11 consolidated case types, which were subsequently audited by human reviewers. Their count remained unchanged during the final three rounds despite continued case-type generation and retention, indicating a within-run plateau.

These results support feedback-guided expansion of long-term memory evaluation requirements. The 11 consolidated case types also exposed failures in MemU and A-Mem through their fixed evaluation cases, supporting cross-system generalization beyond the systems used for discovery. Limitations and further validation are discussed in Appendix H.

AI USE STATEMENT

We used generative AI tools to assist with (i) implementation and debugging of the benchmark pipeline; (ii) generating synthetic benchmark artifacts, including candidate Case Types and

question–answer cases; (iii) translating and polishing manuscript text; and (iv) cleaning and reformatting benchmark data.

The research questions, conceptual framework, hypotheses, benchmark and experimental design, substantive paper narrative, and related-work investigation were developed by the human authors. Human authors also conducted the qualitative and thematic analysis and interpreted all experimental results. This work contains no mathematical claims or proofs for which generative AI assistance was applicable.

Candidate Case Types and evaluation cases were generated automatically during the iterative pipeline, without human intervention in intermediate rounds. Final Type Card admission and consolidation were performed automatically. Human authors conducted a final audit of the resulting types and evaluation cases without modifying the automated assignments. We did not use generative AI to fabricate or alter experimental observations. We reviewed all AI-assisted code, text, data, and benchmark artifacts, and take full responsibility for the final content of this work.

REFERENCES

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation, 2024. URL https://arxiv.org/abs/2402.03216.

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.

Yiming Du, Hongru Wang, Zhengyi Zhao, Bin Liang, Baojun Wang, Wanjun Zhong, Zezhong Wang, and Kam-Fai Wong. Perltqa: A personal long-term memory dataset for memory classification, retrieval, and fusion in question answering. In *Proceedings of the 10th SIGHAN Workshop on Chinese Language Processing (SIGHAN-10)*, pp. 152–164, 2024.

Jiajie Fu, Junwen Chen, Mengzhao Wang, Aoxiang He, Maojia Sheng, Xiangyu Ke, Yifan Zhu, and Yunjun Gao. Vikingmem: A memory base management system for stateful llm-based applications. *arXiv preprint arXiv:2605.29640*, 2026.

Zexue He, Yu Wang, Churan Zhi, Yuanzhe Hu, Tzu-Ping Chen, Lang Yin, Ze Chen, Tong Arthur Wu, Siru Ouyang, Zihan Wang, et al. Memoryarena: Benchmarking agent memory in interdependent multi-session agentic tasks. *arXiv preprint arXiv:2602.16313*, 2026.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=DT7JyQC3MR.

Douwe Kiela, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, Amanpreet Singh, Pratik Ringshia, et al. Dynabench: Rethinking benchmarking in nlp. In *Proceedings of the 2021 conference of the North American chapter of the Association for Computational Linguistics: human language technologies*, pp. 4110–4124, 2021.

Jiho Kim, Woosog Chay, Hyeonji Hwang, Daeun Kyung, Hyunseung Chung, Eunbyeol Cho, Yohan Jo, and Edward Choi. Dialsim: A real-time simulator for evaluating long-term multi-party dialogue understanding of conversation systems. *arXiv preprint arXiv:2406.13144*, 2024.

Letta. Letta: A framework for stateful agents. https://github.com/letta-ai/letta, 2026. Version 0.6.21. Accessed: September 2026.

Xiang Lisa Li, Evan Zheran Liu, Percy Liang, and Tatsunori Hashimoto. Autobencher: Creating salient, novel, difficult datasets for language models. *arXiv preprint arXiv:2407.08351*, 2024.

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, 2026.

Qin Liu, Jacob Dineen, Yuxi Huang, Sheng Zhang, Hoifung Poon, Ben Zhou, and Muhao Chen. Arenabencher: Automatic benchmark evolution via multi-model competitive evaluation. *arXiv preprint arXiv:2510.08569*, 2025.

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

NevaMind-AI. MemU: An agentic memory framework for llms and ai agents, 2025. URL `https://github.com/NevaMind-AI/memU`. GitHub repository.

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

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Yiting Shen, Kun Li, Wei Zhou, and Songlin Hu. Mem2actbench: A benchmark for evaluating long-term memory utilization in task-oriented autonomous agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8173–8190, 2026.

Haoran Tan, Zeyu Zhang, Chen Ma, Xu Chen, Quanyu Dai, and Zhenhua Dong. Membench: Towards more comprehensive evaluation on the memory of llm-based agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19336–19352, 2025.

Mohammad Tavakoli, Alireza Salemi, Carrie Ye, Mohamed Abdalla, Hamed Zamani, and J Ross Mitchell. Beyond a million tokens: Benchmarking and enhancing long-term memory in llms. *arXiv preprint arXiv:2510.27246*, 2025.

Md Nayem Uddin, Kumar Shubham, Eduardo Blanco, Chitta Baral, and Gengyu Wang. From recall to forgetting: Benchmarking long-term memory for personalized agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 26814–26841, 2026.

Volcengine. OpenViking: An open-source context database for ai agents. `https://github.com/volcengine/OpenViking`, 2026. Version 0.4.12. Accessed: September 2026.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, et al. Livebench: A challenging, contamination-limited llm benchmark. In *International Conference on Learning Representations*, 2025.

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

Di Wu, Zixiang Ji, Asmi Kawatkar, Bryan Kwan, Jia-Chen Gu, Nanyun Peng, and Kai-Wei Chang. Longmemeval-v2: Evaluating long-term agent memory toward experienced colleagues. *arXiv preprint arXiv:2605.12493*, 2026.

Chunqiu Steven Xia, Yinlin Deng, and Lingming Zhang. Top leaderboard ranking= top coding proficiency, always? evoeval: Evolving coding benchmarks via llm. *arXiv preprint arXiv:2403.19114*, 2024.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026a.

Jing Xu, Arthur Szlam, and Jason Weston. Beyond goldfish memory: Long-term open-domain conversation. In *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 5180–5197, 2022a.

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

Xinchao Xu, Zhibin Gou, Wenquan Wu, Zheng-Yu Niu, Hua Wu, Haifeng Wang, and Shihang Wang. Long time no see! open-domain conversation with long-term persona memory. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2639–2650, 2022b.

Z.ai. GLM-5.3-Flash. `https://huggingface.co/zai-org/GLM-5.3-Flash`, 2026. Model card. Accessed: 2026-09-20.

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

A NOTATION

Table 4: Notation used in Case Mining, Evaluation, and Evolution.

Symbol	Meaning
u	A user.
$H_u = \{(x_i, t_i, s_i)\}_{i=1}^{n_u}$	The de-identified multi-session conversation history of user u.
x_i, t_i, s_i	A user-authored utterance, its timestamp, and its session identifier.
$\mathcal{P}_t$	The case type pool used in round t.
$\mathcal{P}_0$	The initial case type pool: single-hop, multi-hop, temporal, and adversarial.
τ	A case type.
$\tau = (m_\tau, d_\tau, g_\tau, v_\tau)$	A case type specification.
m_τ	The memory ability tested by the case type and its inclusion and exclusion boundaries.
d_τ	The required evidence configuration.
g_τ	The rule for generating a question and reference answer from evidence.
v_τ	The rule for verifying the resulting case.
$c = (q, a, E, \tau)$	An evidence-grounded evaluation case.
q, a, E	The question, reference answer, and supporting evidence, where $E \subseteq H_u$.
$\mathcal{C}_t$	The cases constructed by Case Mining in round t.
$\mathcal{H}_t$	The cases classified as hard by Evaluation in round t.
$\mathcal{D}_t$	The cases classified as discriminative by Evaluation in round t.
$\mathcal{F}_t = \mathcal{H}_t \cup \mathcal{D}_t$	The evaluation feedback consumed by Evolution.
$\hat{\tau}$	A candidate case type produced by Evolution.
$\widehat{\mathcal{P}}_{t+1}$	The candidate case types produced in round t.
$\mathcal{P}_{t+1}$	The case type pool after admission, used for Case Mining in round $t+1$.

B CASE TYPE TAXONOMY

The 31 automatically admitted Type Cards were grouped into the following 11 consolidated case types by the fixed consolidation procedure, based on shared evidence requirements and reasoning operations. Each illustrative example specifies a minimal evidence set, question, and reference answer to clarify the evidence requirements and reasoning operation of the corresponding type. These examples are not additional benchmark instances.

CT01: Multi-constraint fact retrieval

Definition. Retrieve the directly stated fact that jointly satisfies all constraints in the query, such as the entity, event, time, amount, purpose, or relation. A match on only a subset of the constraints is insufficient.

Evidence. On 3 March, the user paid Bluebird Studio CNY 2,400 for a kitchen deposit. On 18 March, the user paid Northlight Lab CNY 1,800 for bedroom lighting.

Question. Which company received the CNY 1,800 payment for bedroom lighting on 18 March?

Answer. Northlight Lab.

CT02: Temporal version reconstruction

Definition. Recover the value of an attribute that was valid before, during, or after a recorded change. Later updates must not erase a superseded value when the question asks about the earlier state.

Evidence. In April, the user named the project "Cedar." On 9 June, the user renamed it "Atlas" and subsequently used the new name.

Question. What was the project called immediately before it was renamed "Atlas"?

Answer. Cedar.

CT03: Evidence-grounded numerical reasoning

Definition. Derive an unstated numerical result by applying an explicit operation to recorded quantities while respecting the requested scope, units, and inclusion conditions.

Evidence. The user logged cycling distances of 12 km on Monday, 8 km on Wednesday, and 15 km on Saturday.

Question. How far did the user cycle on weekdays in total?

Answer. 20 km.

CT04: Set-presupposition validation

Definition. Test whether the membership or cardinality presupposed by a question is supported by the recorded set. If the presupposition is false or does not determine a unique subset, report the mismatch rather than inventing members or selecting an arbitrary subset.

Evidence. The user considered Paris, Berlin, and Rome for the trip; the history records no rule for preferring two of them.

Question. Which two cities did the user consider for the trip?

Answer. The premise is unsupported: the history records three cities and provides no basis for selecting exactly two.

CT05: Complementary evidence synthesis

Definition. Combine mutually compatible observations that contribute different necessary parts of the answer, while preserving the scope, specificity, and qualifications of each observation.

Evidence. In January, the user reported difficulty falling asleep. In February, the user reported waking twice during the night.

Question. What sleep difficulties did the user report across the two months?

Answer. Difficulty falling asleep and waking twice during the night.

CT06: Bounded trajectory reconstruction

Definition. Reconstruct an ordered development within a specified topic and period, distinguishing among intentions, actions, feelings, outcomes, and retrospective evaluations.

Evidence. The user initially planned to rent alone, later viewed shared flats, then chose a shared flat, and afterwards said that it reduced costs but provided less privacy.

Question. How did the user's housing plan and assessment evolve?

Answer. The user moved from planning to rent alone to choosing a shared flat, which was later assessed as cheaper but less private.

CT07: Bounded set reconstruction

Definition. Recover every member of an explicitly delimited set without omitting listed members, adding related items, or making a choice on the user's behalf.

Evidence. The user's original request asked for a comparison of hiking, swimming, and cycling.

Question. Which activities were listed in the original request?

Answer. Hiking, swimming, and cycling.

CT08: Entity–relation binding verification

Definition. Verify which person or entity a fact, designation, or relation belongs to. In particular, information about someone on whose behalf the user asks a question must not be attributed to the user without evidence.

Evidence. The user said, "My mother prefers quiet hotels. Could you recommend one for her trip?"

Question. Does the user prefer quiet hotels?

Answer. The user's preference is unknown. The stated preference for quiet hotels belongs to the user's mother.

CT09: Interaction-occurrence verification

Definition. Determine whether a specified communicative act occurred using a direct record or a response that uniquely entails that act, while distinguishing occurrence, non-occurrence, and insufficient evidence.

Evidence. The user said, "Customer service replied that my refund request had been accepted and the money would be returned within three business days."

Question. Had the user requested a refund?

Answer. Yes. Acceptance of the refund request indicates that the user had submitted it.

CT10: Conflicting-assertion resolution

Definition. Resolve incompatible claims about the same stable fact using explicit correction or confirmation evidence; when no resolving evidence exists, preserve the conflict instead of selecting a value.

Evidence. In one conversation, the user said, "I paid the rental deposit on 8 May; it was CNY 2,000." In a later conversation, the user described the same payment: "The deposit I paid on 8 May was CNY 3,000." Neither statement was identified as a correction, and no receipt was recorded.

Question. When did the user pay the rental deposit, and how much was it?

Answer. The deposit was paid on 8 May. The amount cannot be determined: the two accounts give CNY 2,000 and CNY 3,000, with no evidence resolving the discrepancy.

CT11: Value-evidence sufficiency

Definition. Determine the most specific value supported when the target entity and attribute are clear. Missing values or details must remain unknown rather than being inferred from the absence of a record.

Evidence. The user said, "I visited the gym to ask about membership on 3 June, but I only started training regularly in late June."

Question. On what exact date did the user start training regularly?

Answer. In late June; the exact date is not specified. The date 3 June refers to the membership inquiry, not the start of regular training.

C Case Type Specifications

A case type in MemEvoBench specifies how to identify evidence, construct a question and reference answer, and check whether the resulting case tests the intended memory behavior. This section presents the four initial case types and an illustrative evolved specification.

C.1 Specification Structure

Each case type is represented as $\tau = (m_\tau, d_\tau, g_\tau, v_\tau)$. The four components serve distinct purposes:

- **Target behavior (m_τ).** Defines the memory behavior being tested and distinguishes it from superficially similar tasks.
- **Evidence requirements (d_τ).** Specifies the historical statements and relationships needed to construct a valid case, including relevant competing or potentially misleading evidence.
- **Generation rules (g_τ).** Specifies how to formulate the question and reference answer from the selected evidence without introducing unsupported facts or revealing the answer.
- **Verification rules (v_τ).** Specifies the checks for evidence support, answer validity, and conformity to the intended case type.

These components are natural-language instructions consumed by the construction and validation stages. The distinction between m_τ and v_τ is important: the former states what a case should test, whereas the latter specifies how to check an individual question–answer pair. A factually correct answer is not sufficient if the question can be resolved without the intended evidence relation.

C.2 Initial Case Types

The initial pool contains single-hop, multi-hop, temporal, and adversarial case types, as introduced in the main text. The following specifications describe their operational boundaries in MemEvoBench.

Single-hop

Target behavior. Recall a specific fact directly stated by the user within one session.

Evidence requirements. Identify a concrete statement that uniquely determines the requested entity or attribute. The answer must not require combining information across sessions or comparing temporal states.

Generation rules. Ask a specific question about the selected fact. Identify the subject and requested attribute clearly, and derive the reference answer only from the user's direct statement.

Verification rules. Check that the evidence comes from one session and directly supports a unique answer. Reject questions that reveal the answer or require cross-session integration, temporal reasoning, or external knowledge.

Multi-hop

Target behavior. Combine at least two indispensable pieces of user evidence to resolve one bounded question.

Evidence requirements. Select distinct user statements that jointly determine a common entity set, relation, or task-specific answer. Prefer statements from different sessions or well-separated turns. Repeated mentions of the same fact do not constitute separate reasoning steps.

Generation rules. Construct a question whose complete answer requires the selected statements together. A multipart answer is permitted when its parts serve the same bounded task; unrelated questions must not be combined merely because they concern the same user or topic.

Verification rules. Remove each selected statement in turn and check whether the remaining evidence still completely and uniquely determines the answer. Reject cases containing dispensable evidence, duplicate statements, a single statement that already provides the full answer, or unrelated factual subquestions.

Temporal

Target behavior. Resolve a question using temporal relationships in the user's history.

Evidence requirements. Identify statements for which temporal order, elapsed time, state changes, or relative-time references are necessary to determine the answer. A timestamp attached to an otherwise static fact is insufficient.

Generation rules. Formulate a natural question that requires the identified temporal relationship. Include time expressions when they define the requested scope, but do not add irrelevant dates, session identifiers, or record positions merely to make the question appear temporal.

Verification rules. Check that the evidence supports the temporal inference and that removing the relevant temporal relationship would prevent a complete and unique answer. Reject cases that only recall a static fact independent of that relationship.

Adversarial

Target behavior. Avoid transferring a recorded fact to a closely related but unsupported person, object, role, or relation.

Evidence requirements. Identify a concrete source fact that is highly relevant to the question but does not establish its target premise. The mismatch may concern the person, role, object, location, time, category, relation, or polarity. Arbitrary missing information is not sufficient to construct a case.

Generation rules. Ask a natural factual question containing the unsupported target premise. The draft reference answer identifies what cannot be determined and distinguishes it from the nearby supported fact. Do not construct the mismatch solely by requesting an unrecorded degree of precision or an unstated explanation.

Verification rules. Check the full history for direct support or explicit denial of the target premise. If either exists, the reference answer must follow that evidence; otherwise, it must preserve uncertainty. Missing evidence must not be treated as evidence of non-occurrence. Reject cases with no concrete relationship between the source fact and the target premise.

C.3 An Evolved Case Type

Evolved case types follow the same four-component specification. The example below presents constrained numerical aggregation: selecting records that satisfy all query constraints and combining their values into a total.

Constrained numerical aggregation

Target behavior. Within an account of one past event, select numerical records that jointly satisfy the requested constraints and combine their values into a single total. Both constraint matching and aggregation are necessary.

Evidence requirements. Identify at least two independent numerical statements attached to distinct sources, objects, or batches within the event. The query constraints must select at least two records, while other records provide partially matching alternatives. All required values must be explicitly stated, and the requested total must not already appear in the history. Cases requiring unstated intermediate quantities, external estimates, or calendar calculations fall outside this type.

Generation rules. Ask for the total under the complete set of selection constraints without revealing the component values or result. Construct the reference answer by aggregating every fully matching record and excluding partial matches. If no record fully matches the constraints, the answer must state that the requested total cannot be determined from the recorded evidence, rather than substitute a partially matching value.

Verification rules. Check that each included value is explicitly supported and satisfies every selection constraint. Verify that no matching record is omitted, no distractor is included, and the aggregation is correct. Reject answerable cases that reduce to one matching record or direct recall of an already stated total. For no-match cases, verify that the answer preserves uncertainty.

Algorithm 1 Case Mining and quality control for user u

Require: User history H_u, Case Type pool $\mathcal{P}_t$, context budget B

$t_q \leftarrow \max_i \text{date}(t_i) + 1$ day
for each $\tau \in \mathcal{P}_t$ **do**
  $\mathcal{W} \leftarrow \textsc{SessionWindows}(H_u, B)$
  $\mathcal{D} \leftarrow \bigcup_{W \in \mathcal{W}} \textsc{Plan}(W, m_\tau, d_\tau, g_\tau)$
  $\mathcal{K} \leftarrow \textsc{DeduplicateAndSelect}(\mathcal{D}, \tau)$
  for each $(E, q, \tilde{a}) \in \mathcal{K}$ **do**
    $(E', a', r) \leftarrow \textsc{BlindReAnswer}(H_u, q, \tau)$
    if $r = \textsc{ambiguous}$ **then continue**
    end if
    $z \leftarrow \textsc{Compare}(q, \tilde{a}, a')$
    if $z = \textsc{equivalent}$ **then**
      $(\bar{E}, \bar{a}) \leftarrow (E, \tilde{a})$
    else
      $(\bar{E}, \bar{a}) \leftarrow \textsc{Resolve}((E, \tilde{a}), (E', a'), q, t_q, d_\tau, g_\tau)$
      if $(\bar{E}, \bar{a}) = \perp$ **then continue**
      end if
    end if
    if $\textsc{Verify}(q, \bar{a}, \bar{E}, t_q, v_\tau)$ **and** $\textsc{MetaCheck}(q, \bar{a}, \bar{E}, m_\tau)$ **then**
      retain $(q, \bar{a}, \bar{E}, \tau)$
    end if
  end for
end for

D Case Mining Details and Quality Control

This appendix specifies the operational protocol underlying Case Mining. For a user history $H_u = \{(x_i, t_i, s_i)\}_{i=1}^{n_u}$, we preserve chronological order and session boundaries. When a history exceeds a context budget B (80K characters in the default configuration), we pack complete sessions into successive windows rather than truncating a session mid-dialogue. All cases constructed for user u share the same question date,

$$t_q = \max_i \text{date}(t_i) + 1 \text{ day},$$

which is the day following the latest user utterance in the complete history. Thus, every retained observation precedes the evaluation point, including evidence outside the window in which a candidate was initially proposed. Evidence is restricted to user-authored content: assistant suggestions, hypotheses, and unconfirmed claims cannot establish a reference answer.

Planner protocol and selection. The Planner receives a Case Type's meta definition, discovery rule, and generation rule, together with one history window; it does not receive the Card's verification rule or assistant responses. After exact question de-duplication, we apply semantic deduplication per user–case-type mining task. Candidates are considered redundant only when they target the same answer target and memory operation and substantially overlap in both question semantics and cited evidence. Topically similar candidates with distinct temporal states, evidence relations, or answer scopes are retained. A selector ranks remaining candidates without generating new ones. It prioritizes Type fit, answer uniqueness, evidence sufficiency, absence of leakage or unsupported claims, diversity across sessions and fact patterns, and potential to discriminate memory systems. Candidates failing a subsequent quality check are discarded rather than automatically repaired.

Core operational templates. We present the decision-relevant constraints of each prompt rather than their literal text. Full prompts, JSON schemas, and prompt versions are provided in the supplementary material.

Planner

Inputs. m_τ, d_τ, g_τ, a target count, and one window of user-authored history.
Instructions.

1. Identify one historical target and an evidence configuration allowed by d_τ *before* writing a question or answer.
2. Treat only established user content as factual evidence. Do not turn requests for advice, hypotheses, plans, third-party events, or unconfirmed premises into user facts.
3. Do not use assistant behavior, external knowledge, guesses, or absence from the current window as evidence.
4. Write a self-contained, uniquely scoped, history-dependent question whose answer cannot be inferred from its wording alone. Do not expose source identifiers, session structure, or a dialogue-like retrieval path.
5. Retain only minimal evidence that directly supports the answer or, when required by the Case Type, distinguishes it from a concrete competing fact.
6. Across the batch, vary the evidence relation, historical target, question form, and required memory operation. Do not create near-duplicates by changing only names, dates, or wording.

Return. plans = [{evidence, question, reference_answer}, ...].

Candidate Selector

Inputs. The Case Type meta definition and verification rule, together with generated candidates and their cited user evidence.
Instructions.

1. Rank, but never rewrite or generate, the supplied candidates.
2. Prefer candidates that satisfy the Case Type and have a natural, single-target, uniquely gradable question.
3. Prefer sufficient and non-redundant evidence, without answer leakage, unsupported claims, external dependencies, or assistant dependencies.
4. Maximize diversity across sessions, fact patterns, and reasoning structures among highly ranked candidates.
5. Prefer candidates likely to discriminate memory systems without sacrificing validity.

Return. Every input candidate identifier exactly once, in ranked order.

Independent reference audit. Case Types differ in what a blind re-answer should establish. For Types whose questions are expected to be resolved by historical facts, a `support_proof` audit checks whether the independently reconstructed answer covers all required direct evidence. For Types that test whether a critical premise is actually supported by the user's history, a `premise_audit` checks for direct confirmation or denial of that premise. This distinction prevents a universal support requirement from rejecting legitimate insufficient-history cases, while preventing an absent record from being converted into an unsupported negative claim. Audit modes are assigned once per Case Type from its executable specification.

Blind Re-answer

Inputs. A question, a Case Type label, an assigned audit_mode, and the complete user-authored history. The Planner's evidence and draft answer are not visible.
Instructions.

1. Independently reconstruct an answer and evidence while preserving the question's entity, relation, condition, and scope.

2. Follow the assigned audit_mode: either establish direct support for the answer or audit whether the question's key premise has direct confirmation or denial.
3. Do not use assistant content, external knowledge, unexecuted plans, or a missing record as a factual answer.
4. Complete every indispensable member of an explicitly bounded set. Mark a case as ambiguous only when distinct direct answers cannot be combined under the original wording.
5. Return minimal, non-redundant evidence that jointly supports the answer. Use boundary evidence only to avoid transferring a concrete competing fact to the target.

Return. {answer, answerability_status, evidence_source_rows, uniqueness, alternative_answer, alternative_evidence_source_rows, ambiguity_reason}.

Answer comparison and reference resolution. The answer comparator first determines whether the Planner answer and blindly reconstructed answer are semantically equivalent. Equivalent pairs retain the Planner reference. Only a non-equivalent pair is passed to the Reference Resolver for evidence-grounded adjudication.

Answer Comparator

Inputs. The question, Planner reference answer, and independently reconstructed answer.
Instructions.

1. Compare the two answers only with respect to factual content, answer scope, temporal state, and conclusion.
2. Do not retrieve evidence, re-answer the question, or determine which candidate is better supported.

Return. {verdict: equivalent | different, deviation_code: none | latest_state_mismatch | aggregation_scope_mismatch | fact_value_mismatch | other}.

Reference Resolver

Inputs. A non-equivalent answer pair; the fixed question and question date; d_τ and g_τ; and the cited user-evidence text for the Planner pair $(E, \tilde{a})$ and Re-answer pair (E', a'). The remaining history is not visible.
Instructions.

1. Select the Planner pair or Re-answer pair only if its own cited evidence fully supports a unique answer.
2. Check that the selected answer satisfies the entity, temporal, and aggregation scope required by the question and Case Type.
3. Do not vote by plausibility, retrieve additional evidence, merge partially supported answers, introduce new facts, or rewrite the question.
4. Reject the candidate when neither submitted pair establishes an unambiguous reference.

Return. {decision: select_planner | select_reanswer | reject, final_evidence, final_answer, reason_code}.

Type-specific quality control. The two final checks receive the selected reference but serve different purposes. The Verifier checks whether the case is valid under the executable conditions of v_τ; the Meta Gate checks whether solving it actually requires the behavioral boundary in m_τ. Although the

Verifier may classify an item as repairable, repair and drop outcomes are both excluded from the released benchmark: no automatic case repair is applied after validation.

Verifier

Inputs. The final question, answer, cited user evidence, question date t_q, and verification rule v_τ. The complete history is not visible.
Instructions.

1. Check that the item is self-contained, history-dependent, and uniquely gradable.
2. Check that the answer cannot be recovered from the question alone.
3. Check that the evidence supports the answer, temporal interpretation, and every condition in v_τ.
4. Reject dialogue-like retrieval routes, assistant-dependent premises, unsupported negative claims, and merely topical background evidence.

Return. {verdict: pass | repair | drop, failure}.

Meta Gate

Inputs. The final question, answer, cited user evidence, and meta definition m_τ.
Instructions.

1. Determine whether solving the case genuinely requires the behavior and evidence relation specified by m_τ, rather than merely a topically related historical fact.
2. Verify that each evidence item provides direct answer support or a Type-required boundary distinction.
3. Do not re-answer the question, introduce unseen history, or reassign the item to another Case Type.

Return. {verdict: match | mismatch, reason}.

E Evolution and Admission

E.1 Strategy-specific candidate generation

All strategies consume validated hard and discriminative feedback, but differ in their parent-type constraints and the transformation used to construct a candidate Case Type.

Table 5: Strategy-specific candidate-generation rules.

Strategy	Feedback input	Parent-type requirement	Candidate operation	Not a valid candidate
Reflect	Failure patterns requiring complementary reasoning structures	At least two existing Case Types	Recombine the parent structures into an independently executable contract	A topical merge or a simple extension of one parent
Refine	Failures associated with one parent Case Type	Exactly one existing parent Case Type	Sharpen an underspecified or conflated evaluation boundary	A renaming, topical substitution, or stylistic rewrite
Explore	Recurring cross-user failure patterns not naturally assigned to a parent	No parent required	Induce an uncovered reasoning operation or evidence boundary	An instance-level variant already covered by the current pool

Note: Source-specific entities, values, dates, and system identities are abstracted away in all candidate specifications.

E.2 AUTOMATIC ADMISSION PROTOCOL

Automatic admission determines whether a candidate Case Type may expand the pool used in the next round. Each candidate is processed sequentially against the complete comparison-eligible Case Type Pool available at that point. Candidates promoted earlier in the same round are therefore included in later comparisons.

Table 6: Automatic admission protocol. A candidate becomes promotion-ready only when it is structurally complete and is classified as novel with respect to the current comparison-eligible Case Type Pool.

Check	**Pass condition**	**Outcome otherwise**
Specification validation	The candidate has a valid identifier and complete target behavior, evidence, generation, and verification rules; it also satisfies the strategy-specific parent-type constraint.	REJECTED
Current-pool comparison	The candidate is neither task-isomorphic to nor in conflict with any comparison-eligible Case Type in the current pool.	REJECTED for duplicate or conflict
Boundary determination	The comparison classifies the candidate as *novel*, establishing a separable evaluation boundary.	PENDING for adjacent or refinement relations
Sequential pool update	A promotion-ready candidate is appended to the pool before the next candidate is processed.	The candidate does not expand the pool

Note: "Comparison-eligible" denotes Case Types marked for novelty comparison in the current pool. A promotion-ready candidate is added as a pending Type Card and is used for Case Mining in the subsequent round.

E.3 WORKED ADMISSION TRACES

To illustrate the provenance retained for automatic admission without releasing raw user histories, Table 7 presents one de-identified trace from the full run. The feedback description abstracts the shared evidence structure and omits source-specific entities, values, and utterances.

Table 7: A de-identified admitted candidate trace from Round 1. The candidate was generated from validated feedback and admitted before being used for Case Mining in the subsequent round.

Field	**De-identified record**
Feedback pattern	Validated feedback contained questions in which one target record coexisted with closely related records that differed in a subject, object, attribute, ordinal position, or time constraint. Correct resolution required applying the complete constraint set rather than substituting a near-match.
Strategy and parents	REFLECT; parent Case Types: `adversarial`, `single_hop`, and `temporal`.
Candidate Case Type	`constrained_slot_binding`. The candidate requires selecting the uniquely matching directly stated record from close alternatives and preserving uncertainty when the constraints do not identify a unique match.
Pool comparison	`novel`; schema check and boundary check both passed. The candidate introduced constraint-bound selection among near-matching records, rather than a topical combination of its parents.
Automatic decision	`promotion_ready`.
Pool update	The candidate was appended as a pending Type Card and became available for Case Mining in the subsequent round.

E.4 CLOSED-LOOP EXECUTION

Algorithm 2 summarizes the operational control flow. Candidate generation uses validated feedback from the current round, whereas automatic admission updates the pool sequentially. Rejected or pending candidates are retained as provenance artifacts but do not expand the Case Type Pool used for the next round.

Algorithm 2. Feedback-guided Case Type Pool evolution

Require: Initial Case Type Pool P_0, user histories $\mathcal{H}$, memory systems $\mathcal{S}$, number of rounds T
Ensure: Updated Case Type Pool P_T

for $t = 0, \ldots, T-1$ **do**
  MINECASES$(\mathcal{H}, P_t) \to C_t$
  EVALUATE$(\mathcal{S}, C_t) \to Y_t$
  VALIDATEDHARDORDISCRIMINATIVE$(C_t, Y_t) \to F_t$
  REFLECT$(F_t, P_t) \cup$ REFINE$(F_t, P_t) \cup$ EXPLORE$(F_t, P_t) \to \hat{P}_{t+1}$
  Initialize P_{t+1} as P_t
  for all $\hat{\tau} \in \hat{P}_{t+1}$ in a fixed processing order **do**
    VALIDATECANDIDATE$(\hat{\tau}) \to v$
    COMPARE$(\hat{\tau}, P_{t+1}) \to r$
    if $v =$ PASS **and** $r =$ NOVEL **then**
      Promote $\hat{\tau}$ and append it to P_{t+1}
    else
      Record the admission decision for $\hat{\tau}$
    end if
  end for
end for

A promoted candidate is appended as a pending Type Card and becomes available for Case Mining in the subsequent round. Consolidation follows each pool update for reporting and stability monitoring without changing the Case Type Pool; human audit follows the online loop.

Source-user and cross-user provenance. We trace each of the 31 retained Type Cards to its admitted candidate's feedback references. The users in these references define its source set; this set agrees with both the first admitted and final card records. We classify each of the 655 unique final cases using its original Card's source set before aggregating into the 11 consolidated types. All 247 inducing reference entries resolve to hard or discriminative feedback in the recorded admission round, and every counted case first appears in an evaluation round after its Card was admitted.

Of the 31 Cards, three have all six users in their source set, and one was admitted in round 10 with no subsequent mining round. The remaining 27 are transfer-eligible. Of these, 23 have recorded non-source mining attempts, 22 have at least one completed non-source mining task, and 21 contribute cross-user cases. One Card has only an unfinished non-source attempt; another completed non-source mining but contributed no cross-user cases. The other four eligible Cards were mined only on source users within the observed run. Thus, eligibility does not imply that cross-user mining was executed or completed.

The resulting split is 392 source-user cases and 263 cross-user cases. These are provenance counts within the retained benchmark subset, not counts of correct system answers. Cross-user means outside a particular Card's inducing source set; the same six users participate in the adaptive loop, so this is not an independently held-out new-user evaluation.

Table 8: Provenance of the 31 retained Type Cards. The six users are labeled User 1–User 6; the Source users column lists their numeric labels. R denotes the admission round, while S and C denote the numbers of source-user and cross-user cases, respectively. Execution status: D = at least one completed non-source task; P = a non-source attempt with no completed task; U = eligible with no recorded non-source attempt; — = ineligible.

Type	Original Type Card	R	Source users	S	C	Exec.
CT01	constrained slot binding	1	4	7	11	D

Continued on next page

Table 8 (continued)

Type	Original Type Card	R	Source users	S	C	Exec.
CT01	reworded evidence resolution	4	3, 5	1	5	D
CT01	narrated episode action attribution	9	2, 3, 4, 5, 6	37	0	U
CT02	creation rename tracking	1	4	3	3	D
CT02	period scoped state recall	2	1, 2, 3, 4, 5, 6	27	0	—
CT03	stated number derivation	1	4	2	7	D
CT03	constrained numeric set aggregation	5	5	2	26	D
CT03	incremental accumulation current total recall	9	1, 2, 3, 4, 5, 6	30	0	—
CT04	unsupported presupposition rejection	1	4	6	40	D
CT04	presupposed count surplus mismatch	9	1, 2, 3, 4, 5, 6	31	0	—
CT05	approximate time union recall	2	3	3	8	D
CT05	ongoing condition facet union	7	2, 3, 4	8	3	D
CT06	bounded period behavior summary	3	2	12	8	D
CT06	initial intent course period summary	6	5	0	24	D
CT06	inquiry action trajectory summary	8	1, 6	6	5	D
CT06	intent drift period summary	8	1, 6	9	0	P
CT06	motive feeling shift arc	8	3	2	3	D
CT06	conception adjustment self evaluation trajectory	9	2, 3, 5	19	0	U
CT06	self action mindset shift period summary	9	1	0	0	U
CT07	multi option request enumeration	3	6	6	21	D
CT08	open attribution undetermined	3	1, 3, 5	9	6	D
CT08	unattested designation undetermined	5	1, 6	28	8	D
CT08	recorded pair identity undetermined	8	3, 4, 5, 6	11	7	D
CT08	unattested designation polar verdict	8	5, 6	19	3	D
CT09	self speech act occurrence verdict	5	4, 5	20	17	D
CT09	counterpart response fixed act verdict	7	5, 6	1	30	D
CT10	self variant conflict resolution	5	1, 2, 4, 5	27	10	D
CT10	unsettled variant conflict undetermined	8	2, 4	10	0	D
CT11	self asked unanswered value	7	1, 2, 3	37	18	D
CT11	partial description value undetermined	9	1, 2, 3, 5	19	0	U
CT11	unrecorded response item count	10	2, 3, 4, 5	0	0	—
				Total	**392**	**263**

Case mining on held-out user histories. To examine cross-user applicability, we applied frozen executable specifications of the 11 consolidated types to two held-out users, User 7 and User 8, neither of whom had participated in type discovery or refinement. Without adapting the specifications or conducting further evolution, the pipeline produced 18 cases for User 7 and 16 for User 8 that passed the automatic quality gates, totaling 34 cases across eight of the 11 types.

F Experimental Configuration

Table 9 summarizes the evaluation protocol, the mining budget recorded for the final evolution round, and the settings of the current implementation.

Memory isolation and answering. Each system ingests the same chronologically ordered history for each user. Memory states are isolated across users and systems and remain fixed during evaluation. For Mem0, Graphiti, OpenViking, MemU, and A-Mem, the shared answerer receives the question, its reference date, and evidence assembled from the system's retrieval outputs. It does not receive case-type labels, supporting evidence annotations, or reference answers. Letta uses its native agent interface and an independent copy of the post-ingestion snapshot for each case. Retrieval-count limits do not impose identical context-token budgets across systems.

Additional-system configurations. MemU uses version 1.5.1 with a SQLite storage adapter and the configured DeepSeek-V4-Flash and BAAI/bge-m3 models. Its native retrieval routing may return no evidence. When resource captions are available, evidence is selected cyclically from memory items, categories, and resource captions, preserving the order within each group, with at most ten records in total. Otherwise, items precede categories. Linked source files are not expanded into full histories.

A-Mem uses the official evaluation implementation at revision `0c8039f`. Its 19,896 message-level notes were constructed using DeepSeek-V4-Flash. The reported evaluation uses BAAI/bge-m3 embeddings for both the retrieval corpus and questions. The original ingestion adapter did not restore the evolution counter between batches, so the reported result evaluates the retained note and link state under this configuration. Retrieval uses the native seed-and-link expansion with $k = 10$; the first ten returned records are passed to the shared answerer in native order, retaining duplicates. Both additional systems completed all 655 cases without unresolved execution errors.

Correctness assessment. The judge receives the question date, question, reference answer, and system answer, but not the system identity, retrieved memories, or answers from other systems. An answer is correct when it covers all conclusions required by the question and is semantically consistent with the reference answer. Equivalent paraphrases and date formats are accepted; optional explanatory details need not be reproduced. The judgment preserves the requested temporal precision and the distinction between established, uncertain, and unsupported claims.

Long-history handling. The implementation divides long histories into windows for case construction. When the full history exceeds the automatic quality-check limit, affected candidates are flagged for manual review instead of being treated as having passed all automatic checks. This case-level review is distinct from the review and consolidation of case types.

Termination and convergence. The experiment terminated at round 10 after the consolidated representation remained unchanged for three consecutive rounds (rounds 8–10), satisfying the predefined stability criterion.

Table 9: Experimental configuration of MEMEVOBENCH, including the source corpus profile, evaluation protocol, recorded final-round mining budget, and reference implementation settings. The latter are reported separately from the recorded experimental configuration.

Component	Configuration
Source corpus profile	
User histories	Six de-identified user histories.
Collection period	Eight months, from November 2025 to July 2026.
Query-answer records	11,416 records.
Unique query texts	11,301 after text-level deduplication.
Sessions	1,365 sessions.
Topical coverage	At least three topic categories per history; six corpus-level categories spanning daily life and social interaction, travel planning, financial investment, health consultation, technology products, and history and nature.
Interaction patterns	Focused recurring-topic interactions and broader multi-topic interactions with topic switches and later revisits.
Evaluation protocol	
Evolution rounds	Ten evolution rounds.
Seed case types	Single-hop, multi-hop, temporal, and adversarial.
Systems reported	Mem0, Graphiti, Letta, and OpenViking supply evolution feedback; MemU and A-Mem test case-type generalization on the fixed 655-case subset.
Proposal and shared-answer generation	DeepSeek-V4-Flash, with temperature 0, for Planner draft construction, input-blinded re-answering, and shared answer generation for Mem0, Graphiti, OpenViking, MemU, and A-Mem.

Component	Configuration
Case release, evolution, and judging	GLM-5.3-Flash, with temperature 0, for reference comparison and resolution, semantic candidate selection, Type-specific verification and Meta validation, candidate-Type evolution and admission, and primary and confirmation judging.
Answer fallback	GLM-5.3-Flash is used only when the primary shared answerer fails the required JSON protocol.
Answering interface	Retrieved evidence followed by the shared answerer for Mem0, Graphiti, OpenViking, MemU, and A-Mem; native agent answering for Letta.
Stopping condition	Three consecutive rounds without a newly consolidated case type (satisfied in rounds 8–10).
Recorded mining budget (final round)	
History window limit	80,000 characters.
Generation budget	One generation pass per window, targeting eight candidate cases per pass.
Candidate selection	At most 100 candidates in the ranking pool; at most ten selected cases per user–case-type mining task.
Quality-control batches	An initial batch of 20 candidates, supplemented in batches of five when needed.
Reference implementation settings	
Required package versions	Mem0 (`mem0ai`) 0.1.63; Graphiti (`graphiti-core`) 0.24.3; Letta 0.6.21; OpenViking 0.4.12.
Embedding model	BAAI/bge-m3 for the four evolution systems, MemU, and A-Mem.
Retrieval request limit	Ten results per query for Mem0, Graphiti, and OpenViking; the number actually returned may be smaller.
Letta answering budget	At most eight chaining steps, followed, if needed, by one final answer-generation step without further retrieval.
Letta evaluation tools	Archival memory search and final-answer submission; memory-writing tools are disabled.
Output-length setting	No explicit `max_tokens` value in the shared JSON client by default; service-side limits still apply.

G CONSOLIDATION AND HUMAN AUDIT

Consolidation procedure

After each pool update, a fixed consolidation model receives the admitted Type Cards, including their target memory behaviors, evidence requirements, generation rules, and verification rules. It assigns cards to the same consolidated type only when they share both a reasoning operation and a stable evidence boundary. Cards that differ in temporal state, answer scope, evidence relation, or verification condition remain separate. The procedure uses a fixed model configuration, prompt, and output schema, and does not specify the target number of consolidated types. Its assignments are used for reporting and cross-type analysis only; they do not alter Type Card admission or subsequent case mining.

Human audit of consolidated types

Three reviewers independently audited the final consolidated types. For each type, reviewers assessed whether its assigned Type Cards shared a stable evidence requirement and reasoning operation. Reviewers recorded audit outcomes and reasons for disagreement. This audit was a quality check only: it did not remove Type Cards, change consolidation assignments, or modify the online Case Type Pool.

Sampled case-level quality audit Separately from the audit of consolidated types, we conducted a sampled quality audit of benchmark cases. Cases were sampled using stratification by user and assigned Type Card to avoid concentrating the audit on a small number of users or frequently occurring types. For each sampled case, the reviewers first assessed whether the question was answerable from the recorded history, whether the supporting evidence was sufficient, whether the reference answer was factually correct, and whether the case matched its assigned case type without leaking the

answer. For cases judged valid, the reviewers then assessed whether each memory-system answer was semantically consistent with the reference answer and supporting evidence. Final judgments were determined by a two-of-three majority vote. This sampled audit served as a quality check and did not constitute exhaustive manual validation of all benchmark cases.

Human audit results

Three reviewers independently assessed 220 cases sampled from the 655-case type-analysis subset, with 20 cases per consolidated type and coverage of all six users. Under majority voting, 209 cases (95.0%) satisfied all case-quality criteria. The remaining 11 cases failed at least one criterion concerning evidence sufficiency, reference-answer correctness, or type alignment.

Before adjudication, all three reviewers agreed on case validity for 198 of the 220 cases (90.0%). For the 209 cases judged valid, the reviewers additionally assessed 836 answers from the four systems used during evolution. Final human correctness judgments agreed with the LLM judge on 784 answers (93.8%). Of the 52 disagreements, 32 answers were judged correct by the LLM judge but incorrect by the human reviewers; 20 showed the reverse pattern.

At the type level, all three reviewers endorsed nine of the 11 consolidated types. The remaining two received majority endorsement after independent review.

H LIMITATIONS

Although MemEvoBench produced reusable, executable case types that could be instantiated on both non-source and held-out user histories, case-type discovery and construction of the main benchmark used six user histories, four seed case types, and four feedback systems. Because the discovery data may reflect user-specific topics and interaction patterns, the resulting 11 consolidated case types should not be viewed as an exhaustive taxonomy of memory requirements.

Evaluation on MemU and A-Mem supports cross-system generalization within the tested configurations while reusing the original benchmark histories. Correctness outcomes also depend on answer generation and automated judging, so they do not isolate failures of individual memory mechanisms.

The initial pool and consolidation procedure may affect which types are discovered. The observed three-round plateau is specific to this run and is not a validated general stopping criterion. Future work should repeat evolution with broader user histories, alternative seeds, and model runs to assess how the discovered taxonomy expands and stabilizes.