GraphMemRL: Action-Native Reinforcement Learning for Persistent Graph Memory Construction

Bingcheng Dong **Shenglan Liu*** **Sifan Zhang** **Jirui Tian** **Weitong Wu**
Dalian University of Technology

Abstract

Long-term memory construction for LLM agents is hindered by two mismatches. The first is a **retrieval-objective mismatch**: existing systems usually retrieve memories for writing with a single holistic query derived from the current input. However, memory construction depends on several complementary aspects of the input, and a single query is insufficient to capture these editing needs. The second is an **optimization-granularity mismatch**: long-horizon memory learning is commonly optimized over full trajectories, whereas memory construction proceeds through incremental state transitions. Effective supervision should therefore assess how each local edit changes the memory state. We propose **GraphMemRL**, a graph-based memory construction framework that addresses both mismatches. GraphMemRL introduces edit-oriented query decomposition to retrieve complementary context for memory writing. It also uses action-native state-transition supervision to directly evaluate whether each graph edit improves the evolving memory state. To make long-horizon learning more tractable, GraphMemRL reformulates memory construction as factorized segment-wise optimization. The agent updates a persistent memory graph across segments under limited context budgets, preserving globally accumulated structure without full-history training. Experiments on long-memory benchmarks show that GraphMemRL improves question answering and produces better connected memory structures with more controlled memory growth. The gains are especially strong on multi-hop, temporal, and knowledge-update tasks.

1 Introduction

Large language models (LLMs) have shown strong capabilities in reasoning, planning, and tool use. Although recent long-context models can process substantially longer inputs, increasing the context window alone does not solve the problem of remembering, organizing, and revising information across extended interactions and sessions [1–4]. Recent work therefore equips LLM agents with external memory systems that preserve user preferences, events, and task-relevant facts across turns and sessions [5–9]. Early approaches mostly rely on fixed pipelines for memory writing, compression, and retrieval [10, 11]. More recent systems study agentic or learned memory management, where the model decides how memories should be stored, updated, linked, or reused [12–17].

Current long-term memory systems still face two unresolved mismatches, illustrated in Figure 1(left). The first is a **retrieval-objective mismatch**. Existing systems often retrieve memories for writing with a single similarity-based query derived from the current segment or its summary. The retrieved memories are then selected by a standard Top-k pipeline. It implicitly treats memory writing as a relevance-matching problem, where the goal is to find memories most similar to the current input. In memory construction, the retrieval target is more complex: the agent needs evidence for deciding whether incoming information is new, redundant, conflicting, or should be linked to prior evidence.

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

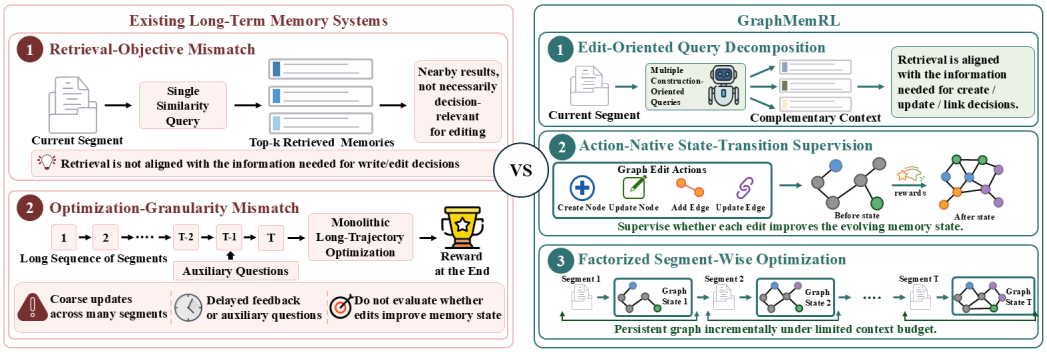


Figure 1: Comparison between existing long-term memory systems and GraphMemRL. Existing methods are limited by retrieval-objective and optimization-granularity mismatches, which make memory writing weakly aligned with memory-state quality. GraphMemRL instead formulates memory construction as incremental optimization of a persistent memory graph through edit-oriented retrieval, action-native state-transition supervision, and segment-wise refinement.

A single holistic query is insufficient for this purpose, since different edit decisions may depend on different aspects of the same observation [18–21]. The limitation becomes more severe in agent memory, where interaction histories are coherent and highly correlated. Naive similarity retrieval can repeatedly return overlapping context and miss the complementary evidence needed for reliable memory editing [22, 23].

The second is an **optimization-granularity mismatch**. Existing long-memory agents are optimized through final task outcomes or task-derived local signals, such as auxiliary question answering, hindsight credit redistribution, or retrieval-centered feedback [12–14, 24, 25]. These signals can improve downstream behavior, yet they indirectly supervise memory construction. They do not answer a central question: did a specific memory action improve the current memory state? Memory learning should provide supervision over the state transition induced by each action. It becomes more important in ultra-long histories, where a single trajectory may contain many segments and thousands of tokens. Monolithic trajectory-level optimization makes credit assignment to local memory edits difficult [13, 14, 26, 27].

To address these two mismatches, we propose **GraphMemRL**, a framework for graph-based long-term memory construction. As shown in Figure 1(right), GraphMemRL consists of three components. First, **edit-oriented query decomposition** rewrites each incoming segment into multiple construction-oriented queries. It allows retrieval to gather complementary context for memory writing. Second, **action-native state-transition supervision** evaluates how each memory edit changes the memory state. The supervision considers both semantic novelty and structural integration, aligning optimization with memory construction quality. Third, **factorized segment-wise optimization** processes long interaction streams segment by segment. The agent operates under a limited context budget and incrementally updates a persistent memory graph across segments. It improves the tractability of long-horizon optimization and preserves globally accumulated memory structure over time.

Experiments on long-memory benchmarks show that GraphMemRL improves downstream question answering and yields memory graphs that are more connected, less redundant, and more controlled in size. The gains are most pronounced on tasks that require multi-hop reasoning, temporal reasoning, and knowledge updates. More broadly, our findings support a state-transition view of memory learning. Long-horizon memory construction can be formulated as a sequence of local edit decisions, with each decision supervised by its effect on the evolving memory state.

Our contributions are threefold:

- We propose **edit-oriented query decomposition** to retrieve complementary evidence for memory writing decisions and reduce redundancy caused by single-query Top-k retrieval.
- We introduce **action-native state-transition utility supervision** to directly evaluate whether each memory edit improves the memory state, beyond task-derived local rewards.

- We formulate long-horizon memory construction as **factorized segment-wise optimization** over persistent graph states, improving tractability under limited context budgets while preserving accumulated memory structure across segments.

2 Related Work

Retrieval and Structured Memory for Agent Memory Some recent work studies how agent memory should be retrieved and organized. This direction includes retrieval-augmented generation, retrieval diversification, temporal memory retrieval, and graph-structured memory representations [7, 27–30]. Standard RAG methods are mainly designed for answer generation over large corpora [19, 20]. Agent memory has a different retrieval setting: the memory store is usually built from bounded, coherent, and highly correlated interaction histories. Naive single-query Top-k retrieval can repeatedly return redundant evidence and fail to expose complementary context [18, 22]. Recent structured-memory systems address this limitation by making retrieval sensitive to temporal or relational organization. For example, Zep and MAGMA incorporate temporal and relational structure into memory retrieval, moving beyond pure semantic similarity [7, 28]. More broadly, graph-based memory studies suggest that relational structure can support long-term reasoning by connecting entities, events, and facts across sessions [29, 31–33]. At the same time, the benefit of graph memory depends strongly on how the graph is constructed and queried. Our work builds on this observation. We study retrieval for memory writing, not only retrieval for answering, and formulate graph construction as a learned sequential decision process guided by action-native utility supervision.

Learning Long-Horizon Memory Management Long-horizon memory for LLM agents has been studied from both system-oriented and learning-based perspectives. Prior work explores persistent memory stores, read–write memory interfaces, structured memory organization, and learned memory control [1, 5–7, 9–11, 16, 17, 34]. Early representative systems such as MemoryBank and MemGPT show that persistent external memory can improve multi-session interaction by storing, retrieving, and updating user-relevant information over time [1, 10]. Subsequent systems further improve the practicality and structure of agent memory. For example, Mem0 emphasizes scalable memory extraction and consolidation, while A-Mem organizes experiences as agentic notes with dynamically maintained links [5, 6]. A growing line of work formulates memory management as a learned decision-making problem. Memory-R1 learns explicit memory operations with reinforcement learning. Mem-T and Fine-Mem introduce denser intermediate supervision to better align memory actions with downstream task success [12–15, 35, 36]. These methods show that memory construction benefits from learning-based control. Our work follows this direction, with a different supervision target. GraphMemRL directly evaluates whether each edit improves the memory state itself. It further optimizes these edits over persistent graph states through factorized segment-wise learning.

3 Method

3.1 Problem Setup

We study long-horizon memory construction for LLM agents. Given an interaction stream

$$H = (h_1, \ldots, h_T),$$

the goal is to incrementally build an external memory that supports future retrieval and downstream reasoning. At segment t, the policy π_θ observes the current segment h_t and accesses the current graph state G_t through the tool interface. It generates a sequence of actions

$$o_t^{(1:K_t)} \sim \pi_\theta(\cdot \mid h_t, G_t),$$

where each action is either a retrieval action or a graph-edit action. The environment then applies the valid graph-edit actions to produce the next graph state:

$$G_{t+1} = \mathrm{Exec}(G_t, o_t^{(1)}, \ldots, o_t^{(K_t)}).$$

3.2 Edit-Oriented Query Decomposition

Memory construction requires retrieval context that can support editing decisions. In GraphMemRL, query decomposition is implemented through the same tool-based action interface used for memory

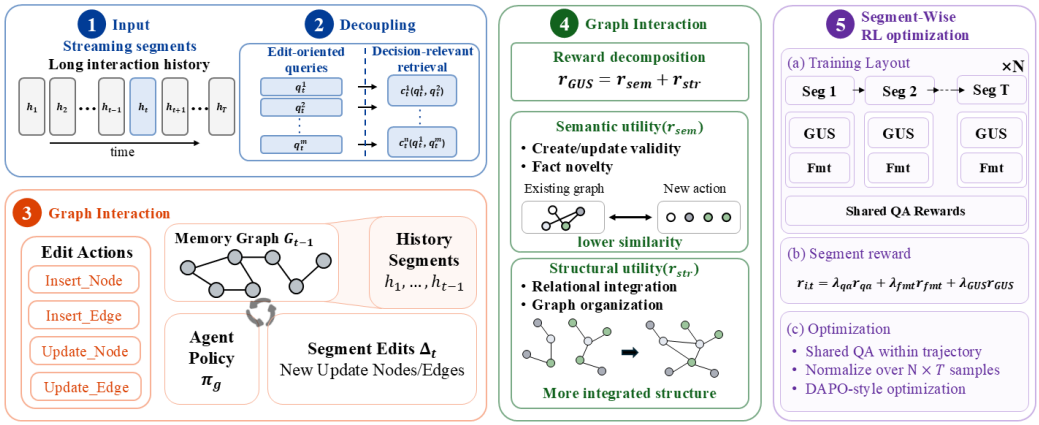


Figure 2: Overview of GraphMemRL. For each input segment, the policy interacts with a persistent memory graph through retrieval and graph-edit actions, including node/edge insertion and update. The resulting graph-state transitions are evaluated by Graph Utility Shaping (GUS), which provides step-wise supervision on edit quality, while the final QA reward is computed by F1 against the reference answer. Training is performed segment by segment with the graph carried forward across the interaction stream, enabling long-horizon memory construction under bounded context.

editing. At each construction segment, the policy is prompted to inspect the existing graph before applying graph edits. Its initial `RETRIEVE` call therefore typically contains a set of construction-oriented queries

$$Q_t = \{q_t^{(1)}, \ldots, q_t^{(m)}\},$$

where each query targets a different editing need.

The graph backend executes these queries and returns complementary graph context

$$C_t = \{c_t^{(1)}, \ldots, c_t^{(m)}\}.$$

Retrieved results from different queries are merged by text span to control context length, while each returned item preserves query-level provenance so that the policy can identify which aspect of the segment it supports. After this initial retrieval, the policy may issue additional `RETRIEVE` actions if the current evidence is insufficient or ambiguous. Thus, edit-oriented query decomposition is a policy-controlled retrieval behavior within the sequential action process, not an external step that precedes the agent.

3.3 Graph Memory and Edit Actions

GraphMemRL represents memory as a labeled graph

$$G = (V, E),$$

where each node $v \in V$ denotes an entity or concept, and each edge $e \in E$ denotes a relation. Nodes and edges maintain associated evidence:

$$v = (\text{name}, \text{type}, F_v), \quad e = (v_{\text{head}}, v_{\text{tail}}, \text{rel}, F_e),$$

where F_v and F_e are sets of time-stamped fact records attached to the node and edge. Each fact is stored as a structured record $\{\text{text}, \text{time}\}$, allowing repeated or evolving observations to be appended without overwriting earlier evidence. The time field enables the model to distinguish older and newer evidence during retrieval and question answering, which is especially important for temporal and knowledge-update queries.

The action space is defined as

$$\mathcal{A} = \{\texttt{RETRIEVE}, \texttt{INSERTNODE}, \texttt{INSERTEDGE}, \texttt{UPDATENODE}, \texttt{UPDATEEDGE}\}.$$

`RETRIEVE` takes one or more short construction-oriented queries as arguments and returns relevant nodes, relations, and local evidence from the current graph. In construction mode, the prompt

encourages the policy to call RETRIEVE before graph editing, while still allowing additional retrieval calls whenever later edit decisions require more evidence. INSERTNODE and INSERTEDGE create new graph elements when the incoming segment contains novel entities or relations. UPDATENODE and UPDATEEDGE append new time-stamped evidence to existing graph elements. Together, these actions allow the agent to expand, consolidate, and revise memory while preserving the historical evidence needed for temporal reasoning.

3.4 Action-Native State-Transition Utility Supervision

We instantiate action-native state-transition supervision with a graph utility shaping signal, denoted as GUS. The purpose of GUS is to evaluate whether a memory edit improves the evolving memory state. Let

$$\Delta_t = (V_t^+, E_t^+, V_t^{\mathrm{upd}}, E_t^{\mathrm{upd}})$$

denote the graph modifications introduced at segment t. Here, V_t^+ and E_t^+ are inserted nodes and edges, and V_t^{upd} and E_t^{upd} are updated nodes and edges. We define the graph utility signal as

$$r_t^{\mathrm{GUS}} = r_t^{\mathrm{sem}} + r_t^{\mathrm{str}},$$

where r_t^{sem} measures semantic utility and r_t^{str} measures structural utility.

Semantic utility evaluates whether the edit contributes useful and non-redundant information to the graph:

$$r_t^{\mathrm{sem}} = w_{\mathrm{val}} r_t^{\mathrm{val}} + w_{\mathrm{fact}} r_t^{\mathrm{fact}}.$$

The term r_t^{val} rewards valid create-versus-update decisions. For example, creating a genuinely new graph element and updating an existing relevant element should receive positive utility. The term r_t^{fact} measures the novelty of newly attached evidence. For a fact f associated with graph element x, its gain is defined as

$$g(f, F_x) = 1 - \max_{f' \in F_x} \cos(e(f), e(f')),$$

where F_x is the pre-edit fact set of x, and $\mathbf{e}(\cdot)$ denotes the text embedding function. A higher value indicates that the new fact adds information that is less redundant with the existing evidence attached to the same graph element.

Structural utility evaluates whether the edit improves graph organization:

$$r_t^{\mathrm{str}} = w_{\mathrm{rel}} r_t^{\mathrm{rel}} + w_{\mathrm{org}} r_t^{\mathrm{org}}.$$

Here, r_t^{rel} evaluates whether new edges integrate evidence into useful graph regions. The term r_t^{org} rewards connected and well-organized graph growth. Overall, GUS encourages edits that add novel evidence, consolidate repeated observations, and improve the structural integration of the memory graph. Implementation-level definitions of GUS, including the reward decomposition used in experiments, are provided in Appendix A.2.

3.5 Policy Optimization

GraphMemRL optimizes the policy over segment-wise decisions while keeping the memory graph persistent across segments. Given an input stream segmented into T steps, we sample a group of N trajectories $\{\tau_i\}_{i=1}^N$ from the same stream. Each trajectory follows the same segmentation and contains a sequence of segment-level memory decisions. The basic optimization unit is therefore (i, t), where i indexes the sampled trajectory and t indexes the segment.

For segment t in trajectory τ_i, the reward is defined as

$$r_{i,t} = \lambda_{\mathrm{qa}} r_{\mathrm{qa},i}^{(t)} + \lambda_{\mathrm{fmt}} r_{\mathrm{fmt},i}^{(t)} + \lambda_{\mathrm{gus}} r_{\mathrm{GUS},i}^{(t)}.$$

Here, $r_{\mathrm{fmt},i}^{(t)}$ is a format reward that encourages valid structured outputs in the required <plan> form. The term $r_{\mathrm{GUS},i}^{(t)}$ is the segment-level graph utility shaping signal computed from the edits made while processing segment t. The QA reward is computed as the token-level F1 score between the final answer and the reference answer. Since all segment-level decisions in the same trajectory

contribute to the final memory state and answer, the same QA reward is assigned to all segments in that trajectory:

$$r_{\mathrm{qa},i}^{(1)} = \cdots = r_{\mathrm{qa},i}^{(T)}.$$

Advantages are computed by normalizing rewards over all $N \times T$ segment-level samples in the rollout group:

$$\hat{A}_{i,t} = \frac{r_{i,t} - \mu_{\mathrm{grp}}}{\sigma_{\mathrm{grp}} + \epsilon}, \qquad \mu_{\mathrm{grp}} = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T} r_{i,t}, \qquad \sigma_{\mathrm{grp}} = \sqrt{\frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T}(r_{i,t} - \mu_{\mathrm{grp}})^2}.$$

We train π_θ with a DAPO-style [37] GRPO objective. We remove the KL-divergence term and use clip-higher for asymmetric clipping:

$$\mathcal{L}(\theta) = -\mathbb{E}_{i,t}\left[\min\left(\rho_{i,t}(\theta)\hat{A}_{i,t}, \mathrm{clip}\left(\rho_{i,t}(\theta), 1-\epsilon_{low}, 1+\epsilon_{high}\right)\hat{A}_{i,t}\right)\right],$$

where

$$\rho_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t} \mid h_t, \mathcal{G}_t)}{\pi_{\theta_{\mathrm{old}}}(o_{i,t} \mid h_t, \mathcal{G}_t)}.$$

During training, the policy is rolled out segment by segment, and the memory graph is carried forward across segments. This formulation makes long-horizon memory construction trainable under limited context budgets. It avoids monolithic full-history rollouts while preserving a global memory state through persistent graph updates. The resulting optimization problem focuses on token-bounded local transitions, with GUS providing direct supervision over edit quality and the final QA reward preserving the downstream task objective. In this way, GraphMemRL learns globally useful memory states through a sequence of locally supervised graph edits.

4 Experiments

4.1 Experimental Setup

Evaluation and Benchmarks. We evaluate GraphMemRL on two long-horizon memory benchmarks: **LoCoMo-10** [2] and **LongMemEval-S** [3]. LoCoMo-10 evaluates long-term conversational memory over multi-session interactions. LongMemEval-S focuses on long-term interactive memory for chat assistants. Following Memory-R1 [12], we adopt the same LoCoMo data configuration with a 1:1:8 train/validation/test split, enabling direct and fair comparison with prior RL-based memory baselines. LongMemEval-S is used only as an additional test benchmark.

Implementation Details. Our primary backbone is **Qwen3-4B-Instruct-2507** [38]. The original VERL framework does not support segment-level expansion of multi-session trajectories with a persistent request-local graph state. We therefore modify the rollout-to-training pipeline so that each segment-level transition can be used as an optimization unit while the corresponding graph state is carried across segments. Additional training configurations are provided in Appendix A.1. Prompt templates, tool schemas, and tool-response formats are summarized in Appendix A.3.

Baselines. We compare GraphMemRL with representative memory baselines reported under the Qwen-2.5-7B-Instruct backbone setting in Memory-R1 [12]. LoCoMo (RAG) [2] serves as a flat retrieval-based baseline. A-Mem and Mem0 [5] represent structured or scalable memory systems, and MemoryOS [39] provides a memory-augmented infrastructure for agent memory. Memory-R1 is an RL-based memory-management baseline. We additionally include GraphTool w/o RL, which uses the same settings as GraphMemRL, but directly runs the backbone model without GraphMemRL training. This baseline tests whether tool access alone is sufficient for reliable graph memory construction.

4.2 Main Results

Results on LoCoMo-10. Table 1 reports LoCoMo-10 results across four question types. GraphMemRL achieves the best average performance among all compared methods, with 47.29 F1, 43.93 B1, and 64.57 judge score. Compared with Memory-R1, GraphMemRL improves the average score

Table 1: LoCoMo-10 results by question type. F1 and BLEU-1 (B1) measure lexical overlap, while J denotes LLM-as-a-Judge accuracy. Higher is better; best results are marked in bold.

Method	Single Hop			Multi-Hop			Open Domain			Temporal			Average		
	F1	B1	J	F1	B1	J	F1	B1	J	F1	B1	J	F1	B1	J
LoCoMo (RAG)	9.57	7.00	15.06	11.84	10.02	19.28	8.67	6.52	12.79	8.35	8.74	5.43	8.97	7.27	12.17
A-Mem	18.96	12.86	40.78	14.73	12.66	31.32	30.58	26.14	46.90	23.67	20.67	28.68	26.08	21.78	40.78
Mem0	24.96	18.05	61.92	20.31	15.82	48.19	32.74	25.27	65.20	33.16	26.28	38.76	30.61	23.55	53.30
MemoryOS	29.55	22.59	48.12	21.03	18.41	38.55	40.85	36.26	63.14	26.26	19.70	24.81	34.64	29.36	51.26
Memory-R1	33.64	26.06	62.34	23.55	20.71	40.96	46.86	38.72	67.81	47.75	30.30	49.61	43.14	33.70	61.51
GraphTool w/o RL	23.31	22.43	47.95	20.41	17.34	35.85	28.63	32.45	52.72	31.06	24.74	38.55	26.12	27.03	47.51
GraphMemRL	**41.79**	**39.87**	**64.75**	**37.52**	**35.62**	**48.67**	**52.45**	**47.83**	**71.47**	**50.23**	**48.73**	**55.42**	**47.29**	**43.93**	**64.57**

by 4.15 F1, 10.23 B1, and 3.06 judge points. The gains are especially pronounced on Multi-Hop questions, where GraphMemRL improves over Memory-R1 by 13.97 F1 and 14.91 B1, suggesting that the learned graph memory better connects evidence distributed across multiple sessions. On Temporal questions, GraphMemRL also improves B1 by 18.43 points and judge score by 5.81 points, indicating stronger handling of time-sensitive and evolving information. GraphTool (no RL) uses the same graph schema, tool interface, and QA-time retrieval protocol as GraphMemRL, but runs the backbone model without reinforcement learning. Despite access to the same graph tools, it substantially underperforms GraphMemRL, with an average gap of 21.17 F1, 16.90 B1, and 17.06 judge points. This result shows that graph-tool access alone is insufficient for reliable long-horizon memory construction. The main benefit comes from learning an action policy that decides when to create, update, and connect graph memories under action-native state-transition supervision.

Table 2: Performance on LongMemEval-S benchmark. Column abbreviations: SSU = Single-session-user, SSP = Single-session-preference, OD = Open Domain, MS = Multi-session, KU = Knowledge-update, TR = Temporal-reasoning.

Method	SSU	SSP	OD	MS	KU	TR	Average
Mem0	54.29	**83.33**	39.29	54.89	75.64	43.61	58.51
Memory-R1	**91.4**	66.7	26.8	63.2	65.4	41.4	59.15
GraphMemRL	81.42	73.33	**42.85**	**66.91**	**78.2**	**45.11**	**64.63**

Results on LongMemEval-S. Table 2 reports the results on LongMemEval-S. GraphMemRL achieves the highest average F1 score of 64.63, outperforming both Mem0 and Memory-R1. The method performs particularly well on **Multi-Session**, **Knowledge Update**, and **Temporal Reasoning**. These categories require the memory system to preserve information across sessions, revise stored facts, and track time-sensitive changes. The gains indicate that persistent graph construction is useful for long-term interactive memory, especially when information is distributed and evolves over time.

4.3 Ablation Study

Figure 3 tracks component-wise training dynamics for Qwen3-4B-Instruct-2507. We report validation answer accuracy at training step 16 for component analysis.

Action-native state-transition supervision improves edit quality. Removing GUS reduces validation ACC from 73.41% to 53.20%, a drop of 20.21 percentage points. The policy still receives the final F1-based QA reward, but Figure 3 shows lower semantic and structural utility, smaller constructed graphs, and more tool execution failures. Together, these observations support the contribution of graph-utility supervision to memory construction beyond the final-answer reward.

Semantic and structural utility provide complementary signals. Semantic Utility only and Structural Utility only achieve 67.15% and 70.41% validation ACC, respectively, both above No GUS but below the full model. Structural Utility only produces larger graphs, averaging 55.91 nodes and 67.50 edges, whereas the full model averages 50.72 nodes and 55.31 edges while reaching 73.41% ACC. Thus, a larger graph does not by itself imply better downstream performance. Combining the two utility components supports a more moderate graph size and higher validation accuracy; these statistics do not independently assess edge correctness.

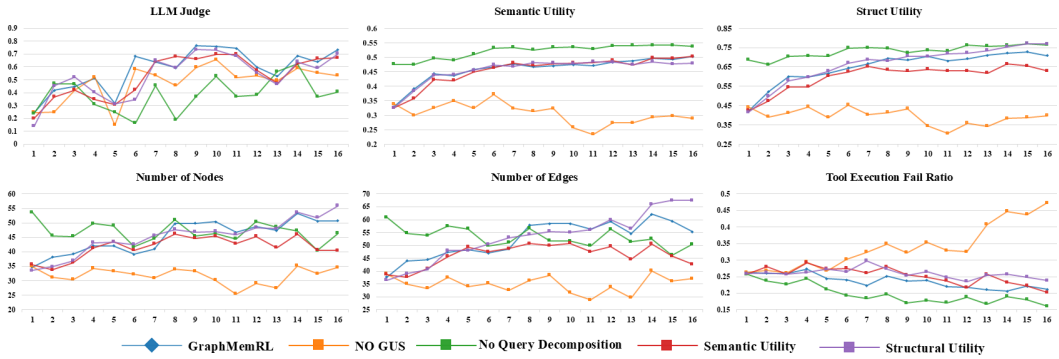


Figure 3: Validation and graph-construction metrics over training steps. **No GUS** removes graph-utility supervision; **No Query Decomposition** retrieves directly from the raw segment; **Semantic Utility** and **Structural Utility** retain only the respective component of GUS. LLM Judge denotes validation answer accuracy. The other panels report semantic utility, structural utility, node and edge counts, and the tool execution failure ratio.

Query decomposition improves retrieval for memory writing. Removing policy-driven query decomposition and retrieving from the raw segment reduces validation ACC to 40.76%. This variant has higher semantic and structural utility scores (0.538 and 0.764) than the full model (0.502 and 0.709), despite its lower answer accuracy. High local utility therefore does not guarantee a memory state that supports downstream QA. The results support retrieving context aligned with editing decisions and interpreting graph-utility scores together with answer performance.

4.4 Training Dynamics and Behavioral Analysis

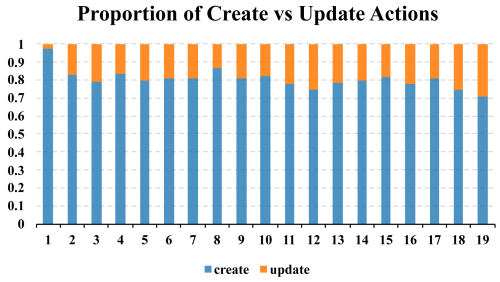


Figure 4: Action distribution across sessions. The update share increases from 2.9% in session 1 to 29.0% in session 19.

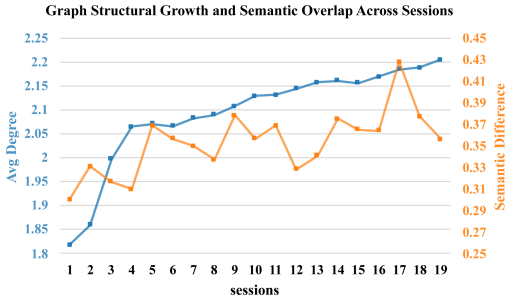


Figure 5: Graph growth and semantic difference across sessions. The left and right axes show average degree and semantic difference, respectively.

We examine how GraphMemRL retrieves context and reuses its persistent graph during memory construction. Using traces from the best-validation Qwen3-4B checkpoint, we retain 2,574 memory-construction segments and exclude the final QA segment from each trajectory. Among segments with retrieval, the policy generates 7.65 queries on average; 63.17% use exactly one retrieval call and 36.83% use multiple calls. The mean pairwise Jaccard overlap between the node sets returned for different queries is 0.117, indicating limited overlap between query-specific results. These observations are consistent with using batched, complementary queries to inspect different aspects of the graph before editing.

Figure 4 shows increasing reuse of existing graph elements. The update-action share rises from 2.9% in session 1 to 29.0% in session 19 as the graph accumulates prior evidence. This shift supports reuse and refinement of persistent memory; action proportions alone do not measure entity-deduplication accuracy.

Figure 5 tracks graph structure and the semantic difference of newly added facts. The edge-to-node ratio increases from 0.910 to 1.099, and average degree rises from 1.818 to 2.204, indicating growing connectivity while the graph remains sparse over the observed sessions. Semantic difference has a mean of 0.353 and ranges from 0.300 to 0.428. This statistic describes the distinction between new and existing facts; it does not independently establish factual correctness or semantic coherence.

4.5 Training with Bounded Rollout Context

We further analyze the model-visible context used in long-horizon memory training. For a stage-aware comparison, we estimate the rollout context size as

$$\widehat{C}_{\mathrm{ctx}} = K_{\mathrm{stage}} \times N_{\mathrm{upd}} \times N_{\mathrm{seq/upd}} \times L_{\mathrm{seq}},$$

where K_{stage} is the number of trainable stages included in the estimate, N_{upd} is the number of global training updates, $N_{\mathrm{seq/upd}}$ is the number of model-visible sequences expanded at each update, and L_{seq} is the length of each sequence. We normalize the batch size to 32 for all methods to focus on sequence-level context rather than batch-size choices.

Table 3: Estimated rollout context size for long-horizon memory training. We use the stage-aware estimate $\widehat{C}_{\mathrm{ctx}} = K_{\mathrm{stage}} \times N_{\mathrm{upd}} \times N_{\mathrm{seq/upd}} \times L_{\mathrm{seq}}$, with a normalized batch size of 32.

Method	Counted training scope / state	$K_{\mathrm{stage}} \times N_{\mathrm{upd}}$	$N_{\mathrm{seq/upd}}$	L_{seq}	$\widehat{C}_{\mathrm{ctx}}$
Memory-R1	Memory manager + answer agent over teacher-built bank	2×200	32×1	6.1k	78.6M
Mem-T	Retrieval MoT over extended observation context	$\geq 1 \times 200$	32×1	40.9k	$\geq$ 262.1M
GraphMemRL	Construction + QA over online persistent graph	1×20	32×28	11.0k	197.1M

Table 3 gives a stage-aware estimate of model-visible rollout context. The estimate is not a complete accounting of all training computation, since prior systems include teacher-assisted preprocessing or separately optimized stages that are not uniformly reported. Memory-R1 has a smaller estimated context size, but its long-range state is provided by a teacher-built memory bank and its memory manager and answer agent are trained separately. Mem-T uses a much longer observation context for MoT-based credit assignment, with additional construction and retrieval optimization stages.

GraphMemRL follows a different formulation. The sequences include both memory-construction segments and the final QA segment within a single segment-wise rollout. These sequences are not concatenated into a full-history trace; cross-session information is carried by the persistent graph. Thus, GraphMemRL is not optimized for the smallest estimated context size, but for training long-horizon memory construction under locally bounded sequences and online persistent state.

5 Conclusion

We presented GraphMemRL, a framework for long-horizon memory construction in LLM agents. GraphMemRL addresses retrieval-objective and optimization-granularity mismatches by combining edit-oriented query decomposition, action-native state-transition utility supervision, and segment-wise optimization over a persistent graph state. It allows the agent to retrieve complementary context for memory writing, directly supervise local graph edits, and carry accumulated memory across long interaction streams under bounded rollout context. Experiments on LoCoMo-10 and LongMemEval-S show that GraphMemRL improves long-term memory question answering, with clear gains on multi-hop, temporal, and knowledge-update tasks. Further analyses show that GraphMemRL builds more connected memory graphs, encourages reuse of existing graph elements, and maintains controlled graph growth. While a fully controlled comparison across long-horizon memory-training paradigms remains an important direction for future work, our results support a state-transition view of memory learning: effective memory agents should not only store and retrieve facts, but also learn how each local edit improves the persistent memory state used by future reasoning.

Acknowledgments and Disclosure of Funding

This work was supported by the National Natural Science Foundation of China under Grant 62376052.

References

[1] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory, 2023. URL https://arxiv.org/abs/2305.10250.

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

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

[4] Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of ai agents, 2026. URL https://arxiv.org/abs/2512.13564.

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

[6] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025. URL https://arxiv.org/abs/2502.12110.

[7] Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory, 2025. URL https://arxiv.org/abs/2501.13956.

[8] Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Simplemem: Efficient lifelong memory for llm agents, 2026. URL https://arxiv.org/abs/2601.02553.

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

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

[11] Ali Modarressi, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. Ret-llm: Towards a general read-write memory for large language models, 2024. URL https://arxiv.org/abs/2305.14322.

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

[13] Yanwei Yue, Boci Peng, Xuanbo Fan, Jiaxin Guo, Qiankun Li, and Yan Zhang. Mem-t: Densifying rewards for long-horizon memory agents, 2026. URL https://arxiv.org/abs/2601.23014.

[14] Weitao Ma, Xiaocheng Feng, Lei Huang, Xiachong Feng, Zhanyu Ma, Jun Xu, Jiuchong Gao, Jinghua Hao, Renqing He, and Bing Qin. Fine-mem: Fine-grained feedback alignment for long-horizon memory management, 2026. URL https://arxiv.org/abs/2601.08435.

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

[16] Ke Yang, Zixi Chen, Xuan He, Jize Jiang, Michel Galley, Chenglong Wang, Jianfeng Gao, Jiawei Han, and ChengXiang Zhai. Plugmem: A task-agnostic plugin memory module for llm agents, 2026. URL https://arxiv.org/abs/2603.03296.

[17] Andy Nguyen, Danh Doan, Hoang Pham, Bao Ha, Dat Pham, Linh Nguyen, Hieu Nguyen, Thien Nguyen, Cuong Do, Phat Nguyen, and Toan Nguyen. Byterover: Agent-native memory through llm-curated hierarchical context, 2026. URL https://arxiv.org/abs/2604.01599.

[18] Zhanghao Hu, Qinglin Zhu, Di Liang, Hanqi Yan, Yulan He, and Lin Gui. Beyond rag for agent memory: Retrieval by decoupling and aggregation, 2026. URL https://arxiv.org/abs/2602.02007.

[19] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection, 2023. URL https://arxiv.org/abs/2310.11511.

[20] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. Raptor: Recursive abstractive processing for tree-organized retrieval, 2024. URL https://arxiv.org/abs/2401.18059.

[21] Saroj Mishra, Suman Niroula, Umesh Yadav, Dilip Thakur, Srijan Gyawali, and Shiva Gaire. Sok: Agentic retrieval-augmented generation (rag): Taxonomy, architectures, evaluation, and research directions, 2026. URL https://arxiv.org/abs/2603.07379.

[22] Tongyu Wen, Guanting Dong, and Zhicheng Dou. Smartsearch: Process reward-guided query refinement for search agents, 2026. URL https://arxiv.org/abs/2601.04888.

[23] Mingfei Lu, Mengjia Wu, Feng Liu, Jiawei Xu, Weikai Li, Haoyang Wang, Zhengdong Hu, Ying Ding, Yizhou Sun, Jie Lu, and Yi Zhang. Choosing how to remember: Adaptive memory structures for llm agents, 2026. URL https://arxiv.org/abs/2602.14038.

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

[25] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners, 2024. URL https://arxiv.org/abs/2308.10144.

[26] Shuai Zhen, Yanhua Yu, Ruopei Guo, Nan Cheng, and Yang Deng. Hierarchical reinforcement learning with augmented step-level transitions for llm agents, 2026. URL https://arxiv.org/abs/2604.05808.

[27] Alessandra Terranova, Björn Ross, and Alexandra Birch. Evaluating long-term memory for long-context question answering, 2025. URL https://arxiv.org/abs/2510.23730.

[28] Dongming Jiang, Yi Li, Guanpeng Li, and Bingzhe Li. Magma: A multi-graph based agentic memory architecture for ai agents, 2026. URL https://arxiv.org/abs/2601.03236.

[29] Juwei Yue, Chuanrui Hu, Jiawei Sheng, Zuyi Zhou, Wenyuan Zhang, Tingwen Liu, Li Guo, and Yafeng Deng. Hypermem: Hypergraph memory for long-term conversations, 2026. URL https://arxiv.org/abs/2604.08256.

[30] Yuanchen Bei, Weizhi Zhang, Siwen Wang, Weizhi Chen, Sheng Zhou, Hao Chen, Yong Li, Jiajun Bu, Shirui Pan, Yizhou Yu, Irwin King, Fakhri Karray, and Philip S. Yu. Graphs meet ai agents: Taxonomy, progress, and future opportunities, 2025. URL https://arxiv.org/abs/2506.18019.

[31] Chang Yang, Chuang Zhou, Yilin Xiao, Su Dong, Luyao Zhuang, Yujing Zhang, Zhu Wang, Zijin Hong, Zheng Yuan, Zhishang Xiang, Shengyuan Chen, Huachi Zhou, Qinggang Zhang, Ninghao Liu, Jinsong Su, Xinrun Wang, Yi Chang, and Xiao Huang. Graph-based agent memory: Taxonomy, techniques, and applications, 2026. URL https://arxiv.org/abs/2602.05665.

[32] Sen Hu, Yuxiang Wei, Jiaxin Ran, Zhiyuan Yao, Xueran Han, Huacan Wang, Ronghao Chen, and Lei Zou. Does memory need graphs? a unified framework and empirical analysis for long-term dialog memory, 2026. URL https://arxiv.org/abs/2601.01280.

[33] Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Xinyu Pan, Zhen Tan, Yu Wang, Tianxin Wei, Shanglin Wu, Ruiyao Xu, Liangwei Yang, Rui Yang, Wooseong Yang, Chin-Yuan Yeh, Hanrong Zhang, Haozhen Zhang, Siqi Zhu, Henry Peng Zou, Wanjia Zhao, Song Wang, Wujiang Xu, Zixuan Ke, Zheng Hui, Dawei Li, Yaozu Wu, Langzhou He, Chen Wang, Xiongxiao Xu, Baixiang Huang, Juntao Tan, Shelby Heinecke, Huan Wang, Caiming Xiong, Ahmed A. Metwally, Jun Yan, Chen-Yu Lee, Hanqing Zeng, Yinglong Xia, Xiaokai Wei, Ali Payani, Yu Wang, Haitong Ma, Wenya Wang, Chenguang Wang, Yu Zhang, Xin Wang, Yongfeng Zhang, Jiaxuan You, Hanghang Tong, Xiao Luo, Xue Liu, Yizhou Sun, Wei Wang, Julian McAuley, James Zou, Jiawei Han, Philip S. Yu, and Kai Shu. Rethinking memory mechanisms of foundation agents in the second half: A survey, 2026. URL https://arxiv.org/abs/2602.06052.

[34] Yupeng Huo, Yaxi Lu, Zhong Zhang, Haotian Chen, and Yankai Lin. Atommem : Learnable dynamic agentic memory with atomic memory operation, 2026. URL https://arxiv.org/abs/2601.08323.

[35] Ziming Wang, Xiang Wang, Kailong Peng, Lang Qin, Juan Gabriel Kostelec, Christos Sourmpis, Axel Laborieux, and Qinghai Guo. Allmem: A memory-centric recipe for efficient long-context modeling, 2026. URL https://arxiv.org/abs/2602.13680.

[36] Xiaohui Zhang, Zequn Sun, Chengyuan Yang, Yaqin Jin, Yazhong Zhang, and Wei Hu. Actmem: Bridging the gap between memory retrieval and reasoning in llm agents, 2026. URL https://arxiv.org/abs/2603.00026.

[37] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

[38] Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

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

A Additional Experimental Details

This appendix provides additional implementation details, reward definitions, and prompting contracts for GraphMemRL. We first describe the training setup, then present the implementation-level definition of Graph Utility Shaping (GUS), and finally summarize the prompt and tool interface used in the reported experiments.

A.1 Experimental Setup

Our main GraphMemRL experiment uses **Qwen3-4B-Instruct-2507** as the policy backbone and adopts a **DAPO-style GRPO** training setup with full-parameter optimization. The main training configuration is summarized in Table 4.

Table 4: Training configuration of the main GraphMemRL run.

Configuration	Value
Optimizer	AdamW
Learning rate	1×10^{-6}
Betas	$(0.9, 0.999)$
Weight decay	0.01
Training framework	FSDP2 with offload
Gradient checkpointing	Enabled
Remove-padding	Enabled
Rollout backend	Asynchronous SGLang
Tensor parallel size	4
Rollout workers	32
Sampling temperature	1.3
Prompts per rollout batch	8
Samples per prompt	8
Prompt token budget	3572
Response token budget	8192
Tool-response token cap	1024
Explicit KL regularization	Disabled
LLM Judge for evaluation	GPT-4.1-mini

Training is conducted on a single node with 4 $\times$ NVIDIA A100 80GB PCIe GPUs. The main 20-step GraphMemRL training run takes approximately one day on this 4-GPU node, corresponding to roughly 96 A100-80GB GPU-hours. Ablation runs use the same hardware configuration and comparable rollout settings. These estimates exclude preliminary debugging and pilot runs.

A.2 Graph Utility Shaping

Graph Utility Shaping (GUS) provides dense segment-level supervision for graph memory construction. For each segment h_t, we compute the reward from the graph modifications induced by the policy. Let

$$\Delta_t = \big(\mathcal{V}_t^+, \mathcal{E}_t^+, \mathcal{V}_t^{\mathrm{upd}}, \mathcal{E}_t^{\mathrm{upd}}\big),$$

where $\mathcal{V}_t^+$ and $\mathcal{E}_t^+$ denote newly created nodes and edges, and $\mathcal{V}_t^{\mathrm{upd}}$ and $\mathcal{E}_t^{\mathrm{upd}}$ denote updated nodes and edges. The segment-level graph utility reward is defined as

$$r_{\mathrm{GUS}}^{(t)} = r_{\mathrm{sem}}^{(t)} + r_{\mathrm{str}}^{(t)}.$$

In the reported implementation, we instantiate this reward as

$$r_{\mathrm{GUS}}^{(t)} = 0.10\, r_{\mathrm{create}}^{(t)} + 0.10\, r_{\mathrm{update}}^{(t)} + 0.15\, r_{\mathrm{node}}^{(t)} + 0.15\, r_{\mathrm{edgefact}}^{(t)} + 0.20\, r_{\mathrm{edge}}^{(t)} + 0.15\, r_{\mathrm{conn}}^{(t)} + 0.15\, r_{\mathrm{cent}}^{(t)}.$$

The first four terms measure semantic utility, including edit validity and fact novelty. The last three terms measure structural utility, including relation integration, connectivity, and centrality.

Edit validity. Edit validity evaluates whether the policy selects appropriate create and update actions. We distinguish successful graph modifications from attempted actions. The sets V_t^+, E_t^+, V_t^{upd}, and E_t^{upd} record successful modifications to the graph. In contrast, $\mathcal{C}_t$ and $\mathcal{U}_t$ denote the attempted create and update actions at segment t, including both successful and failed tool executions. Create validity is computed over attempted create actions:

$$r_{\mathrm{create}}^{(t)} = \mathrm{safeavg}_{x \in \mathcal{C}_t}\, s_{\mathrm{create}}(x),$$

where

$$s_{\mathrm{create}}(x) = \begin{cases} 1, & x \notin G_t \text{ and creation succeeds}, \\ -\alpha_c, & x \in G_t, \\ -\beta_c, & x \notin G_t \text{ but creation fails}. \end{cases}$$

Update validity is defined analogously over attempted update actions:

$$r_{\mathrm{update}}^{(t)} = \mathrm{safeavg}_{x \in \mathcal{U}_t}\, s_{\mathrm{update}}(x),$$

with

$$s_{\mathrm{update}}(x) = \begin{cases} 1, & x \in G_t \text{ and update succeeds}, \\ -\alpha_u, & x \notin G_t, \\ -\beta_u, & x \in G_t \text{ but update fails}. \end{cases}$$

In our implementation, we use $\alpha_c = \alpha_u = 0.25$ and $\beta_c = \beta_u = 0.20$.

For numerical stability, we define

$$\mathrm{safeavg}_{x \in S}\, a(x) = \begin{cases} \frac{1}{|S|} \sum_{x \in S} a(x), & |S| > 0, \\ 0, & |S| = 0. \end{cases}$$

Fact novelty. Fact novelty measures whether newly attached evidence adds information beyond the existing facts in the graph. For a new fact f and a reference fact set $\mathcal{F}$, we define

$$g(f, \mathcal{F}) = \begin{cases} 1 - \max_{f' \in \mathcal{F}} \cos(\mathbf{e}(f), \mathbf{e}(f')), & |\mathcal{F}| > 0, \\ 1, & |\mathcal{F}| = 0. \end{cases}$$

followed by clamping to $[0, 1]$. Here, $\mathbf{e}(\cdot)$ denotes the embedding function. In our implementation, dense retrieval and GUS novelty scoring use the same external `Qwen3-Embedding-4B` backend.

Node-fact gain is computed as

$$r_{\mathrm{node}}^{(t)} = \frac{1}{|\mathcal{F}_{t,\mathrm{node}}^+|} \sum_{(v,f) \in \mathcal{F}_{t,\mathrm{node}}^+} g\big(f, \mathcal{F}_v^{\mathrm{ref}}\big),$$

where $\mathcal{F}_v^{\mathrm{ref}}$ is the reference fact set for node v. For a created node, the reference set is the union of all pre-existing node facts in the graph. For an updated node, the reference set is the pre-existing fact set of the target node.

Edge-fact gain is defined in the same way:

$$r_{\mathrm{edgefact}}^{(t)} = \frac{1}{|\mathcal{F}_{t,\mathrm{edge}}^+|} \sum_{(e,f) \in \mathcal{F}_{t,\mathrm{edge}}^+} g\big(f, \mathcal{F}_e^{\mathrm{ref}}\big).$$

For created edges, the reference set is the union of all pre-existing edge facts. For updated edges, the reference set is the pre-existing fact set of the target edge.

Relation integration. Relation integration rewards new edges according to how they attach evidence to the existing graph. We compute an endpoint-importance term:

$$q(e) = \frac{1}{2}\left(\frac{1}{1 + d(u)/\bar{d}} + \frac{1}{1 + d(v)/\bar{d}}\right),$$

where $d(\cdot)$ is the degree in the pre-edit graph and $\bar{d}$ is the corresponding average degree. The relation-integration score is

$$r_{\mathrm{edge}}^{(t)} = \mathrm{safeavg}_{e \in E_t^+}\, \mathrm{clamp}_{[0,1]}\, q(e).$$

Graph organization. Graph organization contains connectivity and centrality terms. Connectivity rewards newly created nodes that are integrated by newly created edges:

$$r_{\text{conn}}^{(t)} = \begin{cases} \text{clamp}_{[0,1]}\left(0.7\frac{|\{v\in V_t^+ : \exists e\in E_t^+, v\in e\}|}{|V_t^+|} + 0.3\,\text{clamp}_{[0,1]}\frac{|E_t^+|}{|V_t^+|}\right), & |V_t^+| > 0, \\ 0, & |V_t^+| = 0. \end{cases}$$

Centrality rewards new edges that connect to structurally informative regions of the graph:

$$c(e) = \frac{\max(d(u), d(v))}{d_{\max}}, \qquad \eta(e) = \begin{cases} 1.0, & u \in \mathcal{V}_t \text{ or } v \in \mathcal{V}_t, \\ 0.7, & \text{otherwise}, \end{cases}$$

where $d_{\max}$ is the maximum node degree before the edit. The centrality score is

$$r_{\text{cent}}^{(t)} = \text{safeavg}_{e\in E_t^+}\,\text{clamp}_{[0,1]}\,\eta(e)c(e).$$

All averages in GUS are computed with safeavg. When the pre-edit graph has no edges or has zero average degree, we set the endpoint-importance term $q(e)$ to 0. When $d_{\max} = 0$, we set $c(e) = 0$. For an empty reference fact set $\mathcal{F}$ in the novelty function $g(f, \mathcal{F})$, we set $g(f, \mathcal{F}) = 1$, since the new fact is non-redundant with respect to the empty reference set.

A.3 Prompting, Tool Interface, and Tool Responses

GraphMemRL uses an explicit prompt-and-tool contract during both graph construction and final question answering. All segments share a common system-level instruction, while the user prompt changes according to the segment type. We distinguish three cases: the first session, later sessions, and the final QA step. The first two correspond to graph construction, and the last one evaluates whether the constructed graph supports downstream answering.

Prompting contract. The prompting design has two modes: *session-update mode* and *QA mode*. In session-update mode, the model observes only the current session and must inspect prior memory through graph tools. It should not assume access to earlier raw sessions. In QA mode, the model answers from the constructed graph and may only use retrieval. Across both modes, the model is required to produce a `<plan>...</plan>` block for intermediate reasoning. For final QA evaluation, the externally evaluated response must contain only the answer string.

System prompt (abridged)

```
You are a Knowledge Graph Builder Agent for multi-session conversations.
Modes:
(1) Session update mode:  update the graph from the current session.
(2) QA mode:  answer the final question from the existing graph.
Shared rules:
- Each session prompt is independent.
- You only see the current session text.
- If prior information is needed, inspect the graph through tools.
- Be faithful to the graph and the current session.
Core principles:
- Create nodes only for core entities.
- Prefer fewer nodes and more informative edges.
- Use only the supported tool schema.
- Store node facts and edge facts as structured {"text", "time"} records.
QA mode:
- Use only RETRIEVE.
- Never call memory-editing tools.
- Keep the number of searches small.
- Final answer must contain only the answer itself.
Output format:
- Use <plan>...</plan> before tool use for intermediate reasoning.
- In QA mode, the final externally evaluated response must be answer-only.
```

User prompt templates (abridged)

```
First session:
- Identify only the core entities.
- Search before creating new nodes.
- Use only name, type, and facts.
- Store facts as {"text", "time"} items.
- Create relation-rich edges with fact records rather than summaries.
Later sessions:
- Search existing entities first.
- Reuse existing nodes whenever possible.
- Use UPDATENODE to append new fact items to existing nodes.
- Use INSERTNODE or INSERTEDGE only for genuinely new graph elements.
- Use UPDATEEDGE when an existing edge should receive additional or revised relation
facts.
Final QA:
- Use only RETRIEVE if retrieval is needed.
- Search with short entity names, aliases, dates, or event clues.
- Keep the number of searches small.
- Return only the shortest possible answer string.
```

Structured planning format. Each assistant turn begins with a `<plan>...</plan>` block. In construction mode, the plan identifies the core entities in the current session, the relations among them, and the rationale for the intended graph operations. In QA mode, the plan is used only for intermediate reasoning. The final response returned to the evaluator must be answer-only. Missing planning tags are treated as formatting errors during training.

Canonical tool interface. The tool interface uses a single canonical schema. The fields `facts` and `append_facts` are lists of records with required keys `text` and `time`. Malformed field names and unsupported aliases are rejected. This design reduces ambiguity in tool use and ensures that the policy is trained against a stable graph-editing interface.

Append-style updates. Both `UPDATENODE` and `UPDATEEDGE` are append-style operations. They do not overwrite or delete previously stored records. Instead, they add new time-stamped fact records to an existing node or edge. This allows later segments to refine, qualify, or supersede earlier evidence while preserving the historical record accumulated across sessions. In particular, `UPDATEEDGE` is used when an existing relation receives new evidence or temporal development. The relation is therefore represented as a sequence of fact records attached to the same edge.

QA-time restriction. At final answer time, only `RETRIEVE` is available. Memory-editing tools are disabled. This restriction ensures that downstream evaluation reflects the graph constructed during previous segments. It also prevents the model from modifying memory after seeing the final question. Intermediate reasoning in QA mode may still follow the `<plan>...</plan>` format, but the externally evaluated final response must contain only the shortest answer string.

B Case study

Appendix B presents a representative multi-hop example. Under the same graph-tool framework without GraphMemRL training, Qwen3-4B-Instruct-2507 stores relevant events as separate entries and does not attach them to the correct entity-centered structure. GraphMemRL instead updates the existing subgraph around Evan and links the relevant recipient information to the correct node. This organization enables the model to recover the multi-hop evidence needed for the answer. The case study illustrates that the benefit of GraphMemRL is not limited to storing more relevant facts.

C Limitations

GraphMemRL has several limitations. Our empirical study is limited to LoCoMo-10 and LongMemEval-S, with Qwen3-4B-Instruct-2507 as the main policy backbone. Although these benchmarks cover multi-session, temporal, multi-hop, and knowledge-update settings, they do not

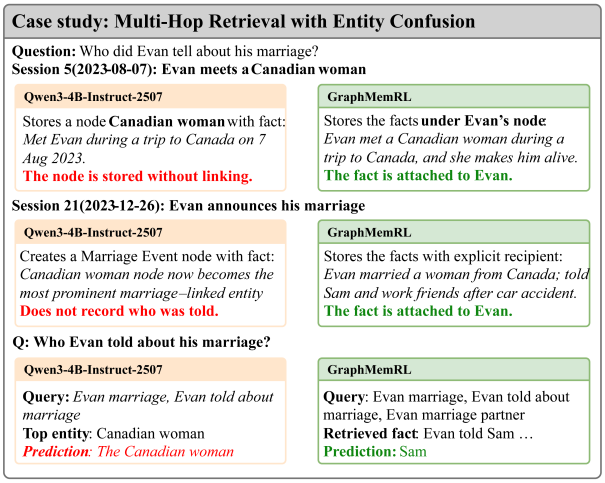


Figure 6: Multi-hop case study: "Who did Evan tell about his marriage?" Under the same graph-tool framework without GraphMemRL training, Qwen3-4B-Instruct-2507 stores relevant events as separate entries and fails to link the recipient information to the correct entity-centered structure. GraphMemRL instead updates the existing subgraph around Evan and attaches the recipient evidence to the relevant node/edge, enabling correct multi-hop retrieval and answer generation.

capture the full diversity of real deployment scenarios. Future work should evaluate GraphMemRL across broader domains, more heterogeneous interaction streams, and additional backbone models.

The current framework also relies on a fixed graph schema and a predefined memory-action interface. This design makes memory construction explicit and controllable, but it may constrain how the agent represents uncertain, conflicting, or rapidly changing information. The current append-style update mechanism preserves historical evidence, but it does not explicitly learn when outdated or incorrect facts should be deleted, hidden, or marked as superseded. Richer mechanisms for forgetting, conflict resolution, and uncertainty-aware revision remain important extensions.

Graph Utility Shaping depends on the chosen utility decomposition, reward weights, and semantic encoder used for fact-novelty estimation. Our ablations show that semantic and structural utilities provide complementary signals, but we do not exhaustively study sensitivity to reward-weight choices, embedding backends, or graph schemas. A more systematic robustness analysis would help clarify how broadly action-native utility supervision generalizes.

GraphMemRL further requires an external graph backend and tool-mediated memory operations during rollout. This makes the memory state persistent and inspectable, but introduces systems-level costs from graph retrieval, tool execution, and state synchronization. Our rollout-context analysis characterizes model-visible sequence length rather than complete end-to-end training cost. Controlled comparisons of runtime, memory usage, and throughput across long-horizon memory-training paradigms remain future work.

Ethical considerations and broader impacts. GraphMemRL studies persistent memory construction for LLM agents. A potential positive impact is that better memory construction can make long-term assistants more reliable, especially when information must be updated, linked, or retrieved across sessions. However, persistent memory systems also introduce privacy and safety risks. If deployed without user control, they may store sensitive user information longer than intended, retrieve outdated or incorrect facts, or expose private information across contexts. Graph-structured memory can further make user profiles more explicit by linking entities, preferences, and events. Responsible deployment should therefore include user-visible memory inspection, consent-based storage, deletion and correction mechanisms, access control, retention limits, and safeguards against using persistent memory for surveillance or unauthorized profiling. Our experiments use existing public benchmarks and do not introduce a new user dataset, but these considerations are important for any real-world deployment of persistent memory agents.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the main contributions of GraphMemRL, including edit-oriented query decomposition, action-native state-transition supervision, and segment-wise persistent optimization. The empirical claims are supported by the experiments and analyses in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed in Appendix C, including benchmark scope, fixed graph schema, reward-design sensitivity, system-level costs.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present theoretical theorems or formal proofs; the contribution is methodological and empirical.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper describes the datasets, data split, backbone model, training framework, rollout setup, prompt budgets, reward definition, tool interface, and evaluation benchmarks in Sections 4 and Appendix A.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The current submission provides detailed implementation, training, and evaluation information, but the code and trained checkpoints are not released at submission time to preserve anonymity. We plan to release the code, reproduction scripts, and trained checkpoints upon acceptance.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4 describes benchmarks, data splits, baselines, and evaluation settings. Appendix A provides training hyperparameters, rollout configuration, reward definitions, prompt templates, and tool interfaces.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We do not report error bars or statistical significance tests due to the computational cost of repeated long-horizon RL training runs. We instead provide ablations, rollout-context analysis, and training-dynamics analyses to support the main findings.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix A.1 reports the GPU type, memory, training framework, rollout backend, and main training configuration. It also provides an approximate wall-clock time and GPU-hour estimate for the main 20-step run.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The study uses public benchmark datasets and focuses on controlled evaluation of long-term memory construction. We discuss limitations and privacy considerations for persistent memory in Appendix C.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The paper discusses potential benefits of more reliable long-term memory agents as well as privacy and safety risks from persistent memory systems in Appendix C.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release a high-risk pretrained model, image generator, or newly scraped dataset. The experiments are conducted on existing benchmark datasets.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper cites the datasets, baselines, backbone model, and training framework used in the experiments. Licenses and terms of use for existing assets are documented in the appendix or supplemental material.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not introduce a new dataset or benchmark asset. The contribution is a training framework and memory-construction method.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or human-subject studies.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or human-subject research.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The paper describes the use of Qwen3-4B-Instruct-2507 as the policy backbone, Qwen3-Embedding-4B for retrieval and novelty scoring, and LLM-based evaluation settings where applicable.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.