# VERMEM: VERIFIER-GUIDED COMMAND-LEVEL CREDIT ASSIGNMENT FOR UNIFIED MEMORY MANAGEMENT

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Long-horizon language model agents must preserve useful knowledge, manage bounded context, and recover earlier task evidence. In existing memory policies that broadcast trajectory-level advantages, the shared signal does not directly assess the semantic quality of individual commands. We introduce VerMem, a unified memory policy with verifier-guided, command-level credit assignment. The local branch evaluates executable memory transitions and normalizes their semantic scores across commands of the same operation type within an update. The global branch normalizes composite feedback across candidate trajectories for the same task. Combining these advantages with direct constraint costs provides differentiated credit while retaining the task-level objective. The policy optimizes complete commands with equal command-level weight and coordinates versioned long-term memory, active context, and same-task history. Supervised initialization and staged reinforcement learning develop these capabilities. Both semantic verifiers provide training feedback and are disabled during evaluation. Experiments on question answering and interactive tasks show improved overall task performance, stronger results from the combined feedback branches, and better performance under constrained online-token budgets.

## 1 INTRODUCTION

Long-horizon language model agents must retain and use task-relevant information across reasoning steps, tool calls, and environment interactions. Agent memory includes stored knowledge and state representations derived from interaction history (Yu et al., 2026; Xiong et al., 2026). Long-term memory (LTM) preserves information for later use, whereas short-term memory (STM) maintains the active context for ongoing reasoning and action (Zhong et al., 2024; Wu et al., 2025; Gao et al., 2025). Retained information must enter this context when needed, requiring coordination between storage and current access.

Existing systems address different parts of this process. Linked notes and extracted facts organize persistent memory (Xu et al., 2025; Chhikara et al., 2025), while structured documents and executive memory support context organization and consolidation (Li et al., 2025; Qian et al., 2026). AgeMem learns LTM and STM operations within one policy (Yu et al., 2026). Earlier observations, tool outputs, and intermediate conclusions may also become relevant after leaving the active context. Same-task recovery differs from persistent retrieval and recent-context summarization. Unified control must preserve these source distinctions.

Learning memory decisions raises a second difficulty. AgeMem uses composite terminal feedback that includes semantic memory signals, and AtomMem learns atomic operations from task-level feedback (Yu et al., 2026; Huo et al., 2026). Shared trajectory advantages assess the complete strategy without directly evaluating each command's semantics. Supported updates can precede unrelated task failures, while redundant retrieval can precede correct answers. Training therefore needs command-specific feedback alongside the delayed task objective.

These requirements define three challenges. **Heterogeneous memory coordination** must preserve the roles of persistent entries, active context, and ordered history within one decision process. **Historical-context recovery** must restore earlier evidence without reintroducing an unbounded his-



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.

tory. **Multi-granularity credit assignment** must distinguish individual transitions while retaining feedback on the complete strategy. Operation utility can depend on later decisions: a supported write may help only after retrieval, while removing necessary context can affect subsequent reasoning.

To address these challenges, we introduce *VerMem*. As shown in Figure 1, one policy coordinates versioned LTM, bounded active context, and same-task history through seven atomic tools and a trainable null command. The local branch evaluates executable memory transitions and compares commands of the same operation type within an update. The global branch compares candidate trajectories for the same task using composite task, evidence, terminal-memory, and efficiency feedback. Their advantages combine with direct constraint costs to train complete commands with equal command-level weight.

Supervised initialization teaches structured commands before staged reinforcement learning develops LTM maintenance, context control, and joint coordination. The curriculum transfers policy parameters between independently initialized task instances. Both semantic verifiers provide training feedback and are disabled during inference.

The method contributes **unified memory control with explicit historical recovery**, **verifier-guided hierarchical credit across distinct comparison sets**, and **complete-command policy learning with equal command-level weight**.

## 2 Related Work

**Memory organization and experience reuse.** Memory organization determines how retained experience returns to use. MemoryBank combines dialogue records, summaries, and user profiles with retrieval and forgetting updates (Zhong et al., 2024); MemGPT uses function calls to move information between limited prompt context and external archival or recall storage (Packer et al., 2023). Experience can also be organized around feedback: CDMem indexes environments and tasks to combine successful procedures with reflections on failures (Gao et al., 2025), while Reflexion stores verbal reflections for subsequent trials without updating model weights (Shinn et al., 2023). VerMem attaches semantic training credit to individual memory transitions.

**Feedback for learned memory control.** Learned memory policies differ in both the object receiving feedback and the alternatives used to normalize it. AgeMem retains LTM across the stages of each candidate rollout and broadcasts composite terminal advantages across memory and reasoning actions (Yu et al., 2026). AtomMem jointly optimizes CRUD and task actions with same-task group-centered returns (Huo et al., 2026). Memory-R1 trains separate memory-management and answer agents from downstream answer rewards, comparing candidate memory operations for the same state (Yan et al., 2026). UMA normalizes query-agnostic memory-session rewards across an update and QA rewards within same-question groups; commands within a session share its advantage (Zhang et al., 2026). VerMem adds direct command evaluation with an operation-specific reference set across the update, combined with same-task trajectory credit. Its loss gives complete memory commands equal weight, and its training phases transfer parameters between independently initialized instances.

**Context consolidation and history access.** Consolidation controls what an agent carries forward. MEM1 jointly learns reasoning and memory consolidation in a compact internal state, optimizing agent-generated tokens along masked trajectories (Zhou et al., 2025). ReSum restarts from query–summary states after periodic history compression; ReSum-GRPO broadcasts the task-group-normalized final-answer advantage to each rollout segment (Wu et al., 2026). MemoBrain trains a dependency-aware copilot with SFT and preference optimization to fold or flush same-task reasoning traces (Qian et al., 2026). VerMem combines context reduction with an explicit action for recovery from ordered same-task history, alongside versioned LTM maintenance and retrieval.

**Process feedback and hierarchical credit.** Intermediate feedback can evaluate different notions of progress. Process-supervised reward models learn labels on mathematical reasoning steps (Lightman et al., 2024), whereas process advantage verifiers measure changes in future success under a prover policy (Setlur et al., 2025). Agent Lightning decomposes trajectories into LLM-call transitions; its reported implementation assigns episode returns to calls, and its interface sup-

ports intermediate rewards (Luo et al., 2025). HiPER uses a plan–execute hierarchy and learned value baselines to credit subgoal switching, subgoal generation, and action execution (Peng et al., 2026). VerMem evaluates executable memory transitions directly and forms local credit through operation-conditioned comparisons. Its frozen semantic verifiers operate during training, distinct from inference-time verification of intermediate decisions and evidence chains in MiroThinker-H1 (MiroMind Team et al., 2026).

## 3 METHOD

VerMem evaluates memory transitions and complete trajectories, then combines their feedback to optimize commands. Figure 1 separates its runtime and training paths.

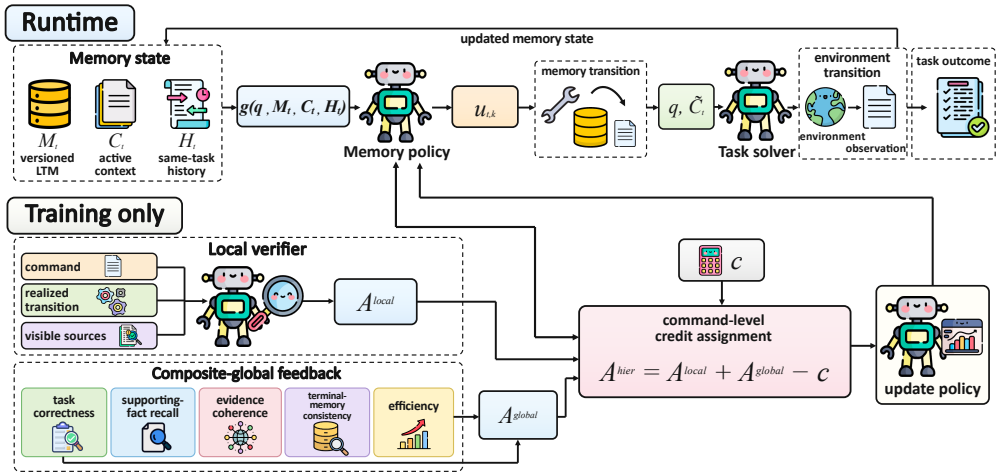



Figure 1: VerMem coordinates memory operations at runtime and learns from local and global feedback during training.

### 3.1 MEMORY STATE AND EXECUTION

For fixed task specification $q$, the memory-decision state and bounded policy input are

$$s_t = (q, M_t, C_t, H_t), \qquad x_t = g(s_t). \tag{1}$$

Versioned LTM $M_t$ retains entries, provenance, revisions, and status. Active context $C_t$ supplies the solver; ordered same-task history $H_t$ retains observations, actions, and tool outputs with provenance. Updates preserve predecessors; deletion is soft. Context removal leaves historical sources recoverable. Unified control coordinates these distinct stores.

The policy selects an operation type $v_t$ and structured arguments $\xi_t$. Executing this complete command gives an intermediate state:

$$u_t = (v_t, \xi_t) \sim \pi_\theta(\cdot \mid x_t), \qquad \widetilde{s}_t = \mathcal{T}_m(s_t, u_t). \tag{2}$$

The seven non-null tools are Add, Update, Delete, Retrieve, Filter, SelectEpisode, and Summarize (Table 5). Retrieve imports active LTM content, whereas SelectEpisode restores fragments from the current task's history. The trainable null command has empty arguments and gives $\widetilde{s}_t = s_t$. Memory decisions have a separate rollout record, so recording null need not append to $H_t$ at this time. Rejected non-null commands preserve $M_t, C_t$ and record the failure and reason in $H_t$; valid non-null commands commit after execution checks.

The task solver acts on the post-memory context, after which environment interaction produces the next decision state:

$$\begin{aligned} b_t &\sim \pi_\phi^{\text{task}}(\cdot \mid q, \widetilde{C}_t), \\ z_{t+1} &\sim \mathcal{P}_{\text{env}}(\cdot \mid b_t), \\ s_{t+1} &= \mathcal{T}_e(\widetilde{s}_t, b_t, z_{t+1}). \end{aligned} \tag{3}$$

The transition appends the action and observation to history and constructs the next active context. Rollouts retain both intermediate and next states; $q$ stays fixed and visible.

The capacities of $C_t$ and $g(s_t)$ are checked independently, even though both are 8,192 tokens. Neither logical object is automatically the complete model prompt. Task-solver calls allow at most 2,048 output tokens; each complete memory command has a 512-token output limit, including its operation name and all structured arguments. Retrieve and SelectEpisode each return at most 1,024 tokens, while also satisfying the capacity of the resulting $C_t$. Summarize produces a shorter faithful representation under the complete-command and active-context limits; it does not use the 1,024-token return cap. These limits govern distinct objects, rather than a single shared allowance for policy input, solver context, and generated output. Appendix A gives the tool schemas and execution requirements.

### 3.2 Command semantics and trajectory utility

The local branch asks whether an executable command makes a justified transition from its current state. The frozen local verifier scores relevance, evidence grounding, local progress, and information fidelity. The four integer scores in $\{0, 1, 2, 3, 4\}$ are divided by four and averaged into $r_{t,k}^{\text{local}}$. Valid nulls have a rubric for leaving memory unchanged; rejected commands bypass scoring. Appendix F.2 defines the dimensions and tool-specific requirements.

The global branch evaluates how the complete memory strategy serves the task. After trajectory $k$ terminates, it receives

$$r_k^{\text{global}} = \frac{r_k^{\text{task}} + r_k^{\text{evid}} + r_k^{\text{state}} + r_k^{\text{eff}}}{4}. \tag{4}$$

The HotpotQA task score is the mean of exact match and token F1, set to zero when no final answer is produced. Evidence quality combines programmatic supporting-fact recall $r_k^{\text{sup}}$ with semantic coherence. The frozen global verifier returns only normalized coherence $v_k^{\text{coh}}$ and terminal-memory consistency $v_k^{\text{state}}$:

$$r_k^{\text{evid}} = \frac{r_k^{\text{sup}} + v_k^{\text{coh}}}{2}, \qquad r_k^{\text{state}} = v_k^{\text{state}}. \tag{5}$$

Coherence concerns final-answer evidence; terminal consistency evaluates active memory against observations. Superseded or failed events remain in history with explicit status. The five signal sources in Figure 1 form four reward components: recall and coherence jointly define evidence quality, alongside task correctness, terminal-memory consistency, and efficiency.

Resource use contributes only when the task score reaches a threshold. Online tokens, task steps, and executed non-null tool calls are separately min–max normalized within the same task's $K$ candidates and averaged into $\overline{C}_k^{\text{online}}$. Components identical across candidates receive zero normalized cost. The efficiency reward is

$$r_k^{\text{eff}} = \mathbb{I}[r_k^{\text{task}} \geq 0.5] \left(1 - \overline{C}_k^{\text{online}}\right). \tag{6}$$

The EM/F1 gate can reward non-exact answers with full token F1. All four terms in Equation (4) lie in $[0, 1]$.

Execution constraints enter credit separately from routine resource consumption. The command cost $c_{t,k}$, capped at 1, covers invalid commands, references, or preconditions; unsupported writes; unjustified deletion; protected-evidence loss; and budget violations. Scoring-format failures do not add command costs. These execution requirements and the verifier's semantic training rubric have different roles (Appendix F).

### 3.3 Hierarchical credit and complete-command optimization

Local and global scores describe different objects and require different comparison sets. Let $\mathcal{B}_{\text{op}}$ contain all memory decisions in the current update and $\mathcal{B}_{\text{exec}}$ contain executable decisions, including valid nulls. For operation type $v$, define $\mathcal{B}_v = \{(q, k, t) \in \mathcal{B}_{\text{exec}} : v_{t,k} = v\}$. The normalized advantages are

$$\begin{aligned} A_k^{\text{global}} &= \text{clip}\left(\frac{r_k^{\text{global}} - \mu_q^{\text{global}}}{\max(\sigma_q^{\text{global}}, 0.01) + \epsilon}, -5, 5\right), \\ A_{t,k}^{\text{local}} &= \text{clip}\left(\frac{r_{t,k}^{\text{local}} - \mu_{v_{t,k}}^{\text{local}}}{\max(\sigma_{v_{t,k}}^{\text{local}}, 0.01) + \epsilon}, -5, 5\right). \end{aligned} \tag{7}$$

Figure 2 separates the two comparison sets. Global statistics compare $K$ trajectories for task $q$; local statistics compare same-type executable commands across the update. Valid nulls form their own operation group. Groups with fewer than eight commands use phase-specific running statistics (Appendix F.4). Both advantages use a 0.01 standard-deviation floor, $\epsilon = 10^{-6}$, and clipping to $[-5, 5]$.

High absolute semantic scores can fall below the operation-group mean. Likewise, a successful trajectory need not have positive global advantage relative to its alternatives. Equal global scores yield zero global advantages. Rejected commands are excluded from local statistics and receive $A^{\text{local}}_{t,k} = 0$; this is not a zero raw score inserted into their operation group.

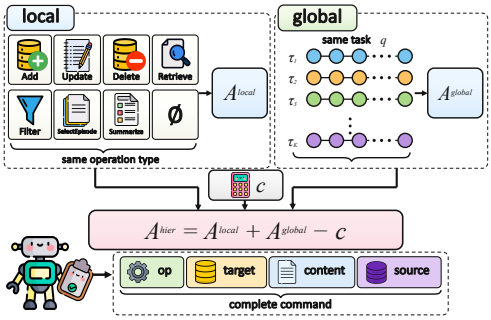



Figure 2: VerMem combines local and global feedback with direct constraint costs to assign credit to complete memory commands.

The local and global advantages and direct cost enter with unit coefficients. Commands share trajectory credit but can differ in local credit. Rejected commands retain global credit and cost; their total need not be negative.

For complete-command tokens $y_{t,k,1:L_{t,k}}$, $\rho_{t,k,j}$ uses the old policy frozen per update; $d^{\text{KL}}_{t,k,j}$ uses a reference fixed per training phase.

The two advantages and the direct cost determine command credit:

$$A^{\text{hier}}_{t,k} = \lambda_{\text{local}} A^{\text{local}}_{t,k} + \lambda_{\text{global}} A^{\text{global}}_{k} - \lambda_{\text{constraint}} c_{t,k}. \tag{8}$$

With PPO clipping and GRPO candidate normalization (Schulman et al., 2017; Shao et al., 2024), we maximize

$$\ell_{t,k,j} = \min\big(\rho_{t,k,j} A^{\text{hier}}_{t,k}, \text{clip}(\rho_{t,k,j}, 1-\epsilon_{\text{clip}}, 1+\epsilon_{\text{clip}}) A^{\text{hier}}_{t,k}\big) - \beta_{\text{KL}} d^{\text{KL}}_{t,k,j},$$
$$\mathcal{J}_{\text{GRPO}}(\theta) = \frac{1}{|\mathcal{B}_{\text{op}}|} \sum_{(q,k,t)\in\mathcal{B}_{\text{op}}} \frac{1}{L_{t,k}} \sum_{j=1}^{L_{t,k}} \ell_{t,k,j}(\theta). \tag{9}$$

The command strip in Figure 2 summarizes argument roles; serialized arguments follow the selected tool's schema in Appendix A.2. Tokens share their command advantage. The inner and outer averages give commands equal weight regardless of argument length; clipping and KL remain token-level. The loss includes arguments, null, and rejected commands, but excludes solver outputs, observations, prompts, and padding. Appendix F.6 defines the token ratios and reference-policy regularization.

### 3.4 Training and Information Boundaries

Supervised initialization teaches structured commands from boundary templates and candidates from a frozen DeepSeek-V3.2 teacher (DeepSeek-AI et al., 2025). All SFT and RL training uses the HotpotQA training split. The teacher sees current evidence, excluding reference answers and future observations. Appendices D.2 and D.3 specify data generation and filtering.

The curriculum then transfers policy parameters through maintenance, context control, and joint coordination:

$$\theta_{\text{SFT}} \to \theta_A \to \theta_B \to \theta_C = \theta^\star. \tag{10}$$

Phase A enables Add, Update, Delete, and null over incremental evidence, delaying solving while $q$ stays visible. Phase B enables Retrieve, Filter, SelectEpisode, Summarize, and null over prepared LTM, distractor-rich context, and same-task history. Phase C enables all commands over complete tasks. Each training instance is initialized independently from its phase-specific memory state. The curriculum transfers policy parameters, not runtime memory states. During evaluation, each episode starts with empty LTM and history.

Each update samples eight tasks and $K = 8$ isolated candidates per task from the same initial state. Null and rejected commands remain in the decision horizon; null is not a non-null tool call. Both

training verifiers use frozen DeepSeek-V3.2 and are disabled during evaluation. The local verifier receives the current bounded state, complete command, realized transition, and visible sources, excluding later observations and the final answer. Global scoring uses completed trajectories and reference labels; policy and solver exclude these labels.

Both scoring roles use at most 128 output tokens, temperature 0, and top-p 1.0. The unique format retry uses the same settings and original scoring material, appending only a format reminder. Formatting failures do not create an additional command constraint cost. Table 9 and Appendix F.1 distinguish the roles' inputs and detail their scoring interfaces.

# 4 EXPERIMENTS

## 4.1 EVALUATION SETTING

We evaluate Qwen2.5-7B-Instruct and Qwen3-4B-Instruct (Qwen et al., 2024; Yang et al., 2025). Evaluation uses held-out HotpotQA distractor validation questions and the reported test splits of ALFWorld, ScienceWorld (SciWorld), AgentBoard PDDL, and BabyAI (Yang et al., 2018; Shridhar et al., 2021; Wang et al., 2022; Ma et al., 2024; Chevalier-Boisvert et al., 2019). The latter four test transfer without additional policy training.

ALFWorld, SciWorld, and BabyAI report success rate (SR), PDDL reports progress rate (PR), and HotpotQA reports $100J_{\text{judge}}$. Average is the arithmetic mean of the five displayed metrics. The held-out `qwen-max-2025-01-25` judge is distinct from the training EM/F1 reward. Figure 3 covers both backbones; Figures 4–7 use Qwen2.5-7B-Instruct. Internal results average seeds 42, 43, and 44. Appendix C specifies identifier-isolated splits, development-based checkpoint selection, and evaluation decoding.

## 4.2 TASK PERFORMANCE AND TRANSFER

Table 1: Task scores under two backbones. ALFWorld, SciWorld, and BabyAI use SR; PDDL uses PR; HotpotQA uses $100J_{\text{judge}}$. Average is the arithmetic mean of the five displayed metrics. Base and external scores are reproduced from Yu et al. (2026). VerMem variants report means over seeds 42, 43, and 44. Bold and underlining identify the highest and second-highest displayed values.

| | | | | | HotpotQA | |
|---|---|---|---|---|---|---|
| **Method** | **ALFWorld** | **SciWorld** | **PDDL** | **BabyAI** | $100J_{\text{judge}}$ | **Average** |
| **Qwen2.5-7B-Instruct** | | | | | | |
| Base | 27.16 | 13.80 | 10.15 | 50.80 | 38.36 | 28.05 |
| LangMem | 38.27 | 28.29 | 15.85 | 51.34 | 37.43 | 34.24 |
| A-Mem | 34.68 | 28.06 | 18.39 | 58.82 | 43.95 | 36.78 |
| Mem0 | 37.49 | 26.99 | 13.96 | 60.58 | 46.66 | 37.14 |
| Mem0$^g$ | 35.34 | 30.50 | 14.86 | 58.78 | 42.06 | 36.31 |
| AgeMem | <u>41.07</u> | <u>35.55</u> | 17.31 | <u>61.42</u> | <u>54.44</u> | <u>41.96</u> |
| VerMem-noVerify | 40.18 | 33.94 | **22.41** | 59.21 | 53.08 | 41.76 |
| **VerMem (Ours)** | **46.30** | **43.39** | <u>22.06</u> | **65.55** | **62.75** | **48.01** |
| **Qwen3-4B-Instruct** | | | | | | |
| Base | 38.51 | 47.89 | 30.14 | 55.83 | 47.48 | 43.97 |
| LangMem | 40.89 | 50.42 | 28.42 | 53.80 | 42.70 | 43.25 |
| A-Mem | 34.31 | 50.14 | 34.41 | 61.35 | 48.48 | 45.74 |
| Mem0 | 41.17 | 51.38 | 31.72 | 60.05 | 39.16 | 44.70 |
| Mem0$^g$ | 36.69 | 47.76 | 29.61 | 57.59 | 38.12 | 41.95 |
| AgeMem | <u>48.97</u> | <u>59.48</u> | <u>35.07</u> | <u>72.56</u> | <u>55.49</u> | <u>54.31</u> |
| VerMem-noVerify | 48.63 | 58.92 | 34.76 | 72.11 | 53.79 | 53.64 |
| **VerMem (Ours)** | **53.73** | **66.37** | **38.75** | **76.77** | **63.61** | **59.85** |

Table 1 reports the full comparison; Figure 3 visualizes full VerMem and its internal task-only control. Full VerMem improves Average by 6.25 and 6.21 points across the two backbones. Nine of ten task comparisons favor full feedback; Qwen2.5 PDDL is the exception (noVerify: 22.41; full: 22.06). Signed values in the figure give full VerMem minus noVerify. The internal control shares

tools, SFT, curriculum, constraints, and GRPO settings with full VerMem, but uses task-outcome feedback; full VerMem adds local and composite-global credit.

Open markers show each task's strongest cited external score. Full VerMem exceeds these scores, with Average differences from AgeMem of 6.05 and 5.54 points; A-Mem is the strongest external reference on Qwen2.5 PDDL. Gains on the four downstream benchmarks support transfer from HotpotQA. Table 1 gives all scores. Its six external rows come from Yu et al. (2026) under their original configurations, including LangMem's online tools and background consolidation (LangChain Team, 2025); these rows are distinct from the internal comparisons.

### 4.3 Performance under online-token budgets

Figure 4 evaluates online-token hard limits from 1,000 to 5,000, using equally weighted macro-average SR across ALFWorld, SciWorld, and BabyAI. These budget-controlled comparisons are separate from the cited main-table rows. Budget exhaustion before completion counts as failure. Appendix C.7 specifies the reported three-seed means and one-standard-deviation bands, together with the budget and accounting protocol.

VerMem leads over the plotted budget range (left). It reaches the target $S^* = 40$ at a reported threshold budget of 2,080 tokens, compared with AgeMem's 2,600, a 20.0% reduction. At the fixed budget $B^* = 2{,}500$ (right), VerMem scores 44.30 SR, exceeding AgeMem by 5.30 percentage points (Table 2). Threshold budget measures the limit needed to reach target performance; TN measures actual consumption.

Table 2: Budget-controlled efficiency with Qwen2.5-7B-Instruct. $\mathrm{TN@}S^*$ is the threshold budget for macro-average SR $S^* = 40.0$; $\mathrm{SR@}B^*$ is macro-average SR at budget $B^* = 2{,}500$.

| Method | $\mathrm{TN@}S^* \downarrow$ | $\mathrm{SR@}B^* \uparrow$ |
|---|---|---|
| A-Mem | 4,250 | 33.80 |
| Mem0 | 3,820 | 35.60 |
| AgeMem | 2,600 | 39.00 |
| **VerMem** | **2,080** | **44.30** |

### 4.4 Cumulative configurations and feedback branches

Figure 5 compares cumulative configurations. From Base, +LT enables LTM tools and +LT/ST enables the full LTM/STM space, both with SFT. The subsequent RL configurations add task-outcome feedback and constraints (+noV), then local and composite-global credit (+V). Scores increase along the sequence on all three tasks. Lines connect these discrete configurations, not training-time observations. The changing tools and training define configuration-level gains rather than isolated component effects; Appendix E gives the endpoints and the published Base reference.

Figure 6 compares feedback branches under shared architecture, initialization, data, constraints, and optimization settings. VerMem-noVerify uses task-outcome feedback and constraints; +Local adds local credit, +Global replaces task-only with composite-global credit, and +Local+Global is full VerMem. Among the single-branch variants in Table 3, +Local leads on ALFWorld and SciWorld and +Global on HotpotQA. Their combination leads on all three. Because +Global combines programmatic evidence and efficiency with semantic verification, these gains concern the joint feedback configuration, not an isolated semantic verifier.

Table 3: Credit-branch ablation. noVerify uses task-outcome advantage and constraints. +Local adds operation-specific local advantage; +Global replaces task-only with composite-global advantage. Their combination is full VerMem. Architecture, SFT initialization, training data, constraints, and GRPO settings are shared.

| **Variant** | **ALFWorld** SR | **SciWorld** SR | **HotpotQA** $100J_{\mathrm{judge}}$ |
|---|---|---|---|
| VerMem-noVerify | 40.18 | 33.94 | 53.08 |
| +Local | 44.72 | 39.81 | 58.26 |
| +Global | 43.31 | 37.96 | 60.18 |
| **+Local+Global** | **46.30** | **43.39** | **62.75** |

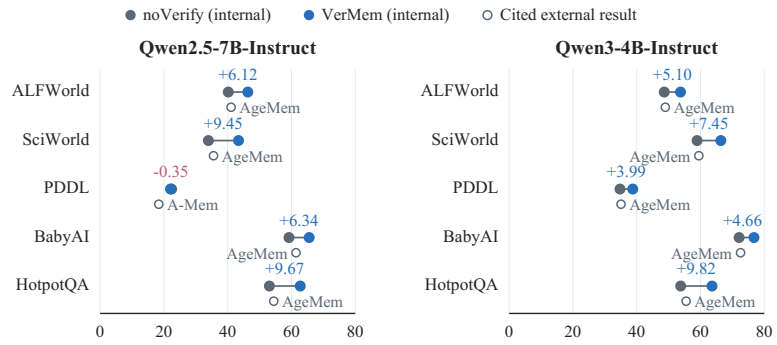



Figure 3: Task performance across two backbones. The horizontal axes show reported task scores; signed values give VerMem minus noVerify, and open markers show cited external results.

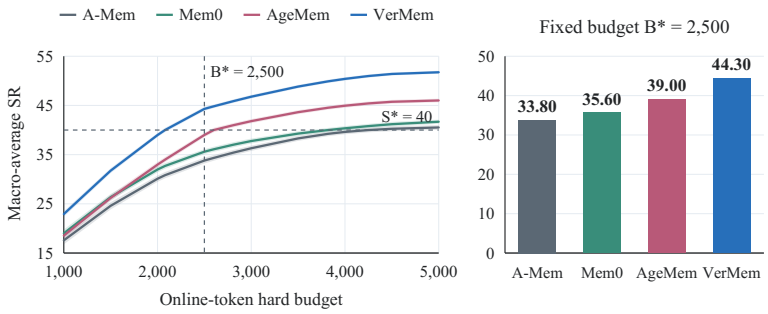



Figure 4: Macro-average success under online-token budgets (left) and at a fixed budget of 2,500 tokens (right).

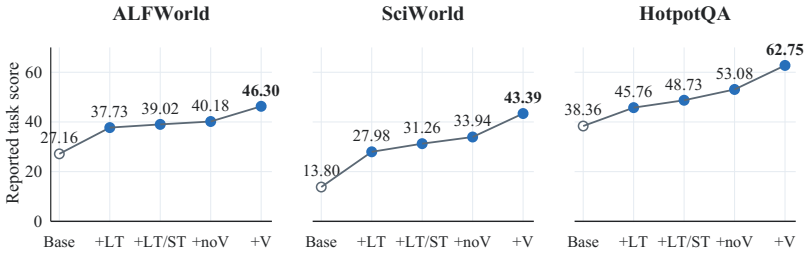



Figure 5: Task performance across cumulative memory and training configurations.

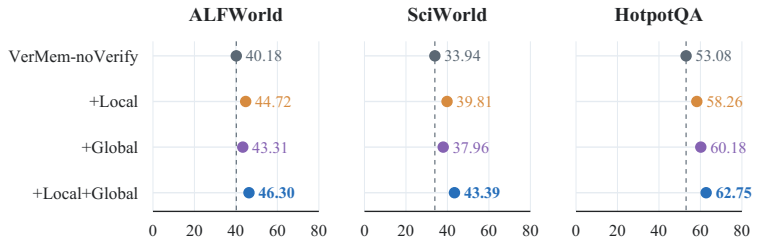



Figure 6: Task performance with local, global, and combined feedback. The horizontal axes show reported task scores; dashed lines mark noVerify.

### 4.5 Reward Design and Training Dynamics

Figure 7 shows the mean Answer-Only and All-Returns monitors during training. Both finish above their initial values, with fluctuations across records. Each panel tracks a strategy-specific monitor rather than a common task-performance metric. Appendix F.7 defines the monitor mappings, recording schedule, and averaging procedure.

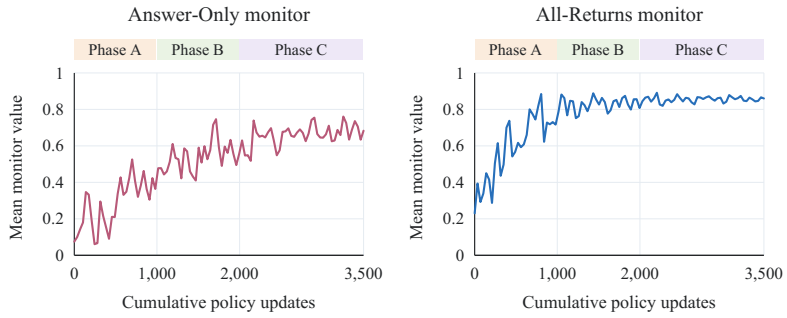



Figure 7: Mean training monitors for Answer-Only and All-Returns on HotpotQA; each panel uses its strategy-specific mapping.

Table 4 compares common held-out metrics. All-Returns improves answer and memory quality while reducing mean actual tokens by 168 (7.14%). Tool calls rise from 4.47 to 5.63; these aggregate changes do not identify which operations cause the token reduction.

Table 4: HotpotQA reward ablation with Qwen2.5-7B-Instruct: judge score $J_{\text{judge}} \in [0, 1]$, mean online tokens (TN), memory quality (MQ), and mean non-null tool calls (TC).

| **Strategy** | $\mathbf{J}_{\text{judge}} \uparrow$ | **TN** $\downarrow$ | **MQ** $\uparrow$ | **TC** |
|---|---|---|---|---|
| Answer-Only | 0.531 | 2,352 | 0.491 | 4.47 |
| **All-Returns** | **0.628** | **2,184** | **0.673** | 5.63 |

The reward comparison additionally measures terminal LTM quality (MQ), actual online tokens (TN), and non-null tool calls (TC). TN counts every online request's inputs and outputs, including repeated inputs, null/rejected tokens, and applicable generative memory or execution-check modules. Offline training, indexing, training verifiers, and judges are excluded (Appendix C).

The monitor definitions explain the separate scales. Answer-Only subtracts the auxiliary cost from the task score. All-Returns combines the trajectory-averaged local score and composite global reward, subtracts the auxiliary cost, and divides by two. Both outputs are clipped to $[0, 1]$. The local average uses the full memory-decision horizon, including null and rejected commands, rather than the operation-specific reference set used for policy advantages. These curves therefore track their respective training signals; neither monitor replaces $A^{\text{hier}}_{t,k}$ in the command objective. The common held-out metrics assess answer quality, retained memory, and online resource use across the two configurations. Checkpoint selection uses the internal development set, separately from these monitors and the held-out judge scores (Appendices B.2 and F.7).

## 5 Limitations

The evaluation covers within-episode memory and transfer from HotpotQA to four downstream benchmarks. Memory is reset between task instances, so these results concern episode-level control rather than continual memory accumulation across tasks. The cumulative comparison varies tools and training jointly; the feedback ablation compares local and composite-global configurations, not isolated effects of SelectEpisode or the semantic global verifier. Cited external results retain their published protocols. The scoring pipeline uses proprietary model snapshots. Online-token accounting excludes offline training, indexing, and evaluation; latency and total computation are separate resource measures.

## 6 Conclusion

VerMem learns unified memory control from the semantics of individual transitions and the utility of complete trajectories. Operation-specific local credit and task-conditioned global credit jointly optimize complete memory commands. The task, budget, and feedback comparisons support this design across the evaluated settings. They connect the quality of individual memory decisions to the outcomes of the strategy those decisions form.

## AI USE STATEMENT

DeepSeek-V3.2 generates SFT-data candidates and supplies the frozen local and global training verifiers. Qwen-Max evaluates held-out answers and terminal memory quality. Their roles and information boundaries are described in Sections 3.4 and 4.1; their configurations and original prompts appear in Appendices C, D, and F.

## ETHICS STATEMENT

The study evaluates memory policies on public question-answering and simulated agent benchmarks, with memory reset between task instances. Reference answers and supporting-fact annotations are confined to the construction, scoring, and evaluation paths specified in Appendix B.3. Teacher, verifier, and judge outputs can contain errors; the empirical claims concern the benchmark settings and evaluation scope described in Section 5.

## REPRODUCIBILITY STATEMENT

Section 3 defines the memory states, execution branches, and command-level credit objective. Appendices A–F describe tool contracts, training-data construction, model roles, capacity limits, prompts, and normalization settings. Section 4.2 separates internal comparisons from cited external scores. Appendix C.7 defines the budget metrics and online-token accounting. Appendix F.7 specifies the training-monitor mappings, recording schedule, and mean aggregation. The experimental tables report the configurations and evaluation metrics for each comparison.

## REFERENCES

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: A platform to study the sample efficiency of grounded language learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJeXCo0cYX.

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.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Erhang Li, Fangqi Zhou, Fangyun Lin, Fucong Dai, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoran Wei, Haowei Zhang, Haowen Luo, Haozhe Ji, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, Jialiang Huang, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jingchang Chen, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexin Huang, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Liang Zhao, Liangsheng Yin, Lihua Guo, Lingxiao Luo, Linwang Ma, Litong Wang, Liyue Zhang, M. S. Di, M. Y Xu, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Panpan Huang, Peixin Cong, Peiyi Wang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, S. H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Songyang Zhou, Tao Ni, Tao Yun, Tian Pei, Tian Ye, Tianyuan Yue, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjun Gao, Wentao Zhang, Xi Gao, Xiangwen Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyuan Li, Xu Chen, Xuecheng Su, Xuehai Pan, Xuheng Lin, Xuwei Fu, Y. Q. Wang, Yang Zhang, Yanhong Xu, Yanru Ma, Yao Li, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yiliang Xiong, Ying He, Ying Zhou, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiao Chen,

Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuduan Wang, Yue Gong, Yuhan Wu, Yuheng Zou, Yukun Li, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehua Zhao, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhiyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Zizheng Pan, Zongqing Yao, Bei Feng, Hui Li, J. L. Cai, Jiaqi Ni, Lei Xu, Meng Li, Ning Tian, R. J. Chen, R. L. Jin, S. S. Li, Shuang Zhou, Tianyu Sun, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xinnan Song, Xinyi Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, Dongjie Ji, Jian Liang, Jianzhong Guo, Jin Chen, Leyi Xia, Miaojun Wang, Mingming Li, Peng Zhang, Ruyi Chen, Shangmian Sun, Shaoqing Wu, Shengfeng Ye, T. Wang, W. L. Xiao, Wei An, Xianzu Wang, Xiaowen Sun, Xiaoxiang Wang, Ying Tang, Yukun Zha, Zekai Zhang, Zhe Ju, Zhen Zhang, and Zihua Qu. DeepSeek-V3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

Pengyu Gao, Jinming Zhao, Xinyue Chen, and Yilin Long. An efficient context-dependent memory framework for LLM-centric agents. In Weizhu Chen, Yi Yang, Mohammad Kachuee, and Xue-Yong Fu (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pp. 1055–1069, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.naacl-industry.80. URL https://aclanthology.org/2025.naacl-industry.80/.

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.

LangChain Team. LangMem. Software library and official documentation, 2025. URL https://langchain-ai.github.io/langmem/. Accessed September 14, 2026.

Rui Li, Zeyu Zhang, Xiaohe Bo, Zihang Tian, Xu Chen, Quanyu Dai, Zhenhua Dong, and Ruiming Tang. CAM: A constructivist view of agentic memory for LLM-based reading comprehension. In *Advances in Neural Information Processing Systems*, volume 38, pp. 113381–113406. Curran Associates, Inc., 2025. doi: 10.52202/085713-3781. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a488aa1a0c00d76db8a922ef7815a786-Abstract-Conference.html.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. Agent Lightning: Train ANY AI agents with reinforcement learning, 2025. URL https://arxiv.org/abs/2508.03680.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 74325–74362, 2024. doi: 10.52202/079017-2365. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/877b40688e330a0e2a3fc24084208dfa-Abstract-Datasets_and_Benchmarks_Track.html.

MiroMind Team, S. Bai, L. Bing, L. Lei, R. Li, X. Li, X. Lin, E. Min, L. Su, B. Wang, L. Wang, L. Wang, S. Wang, X. Wang, Y. Zhang, Z. Zhang, G. Chen, L. Chen, Z. Cheng, Y. Deng, Z. Huang, D. Ng, J. Ni, Q. Ren, X. Tang, B. L. Wang, H. Wang, N. Wang, C. Wei, Q. Wu, J. Xia, Y. Xiao, H. Xu, X. Xu, C. Xue, Z. Yang, Z. Yang, F. Ye, H. Ye, J. Yu, C. Zhang, W. Zhang, H. Zhao, and P. Zhu. MiroThinker-1.7 & H1: Towards heavy-duty research agents via verification, 2026. URL https://arxiv.org/abs/2603.15726.

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. doi: 10.48550/arXiv.2310.08560. URL https://arxiv.org/abs/2310.08560v2. Version 2, February 12, 2024.

Jiangweizhi Peng, Yuanxin Liu, Ruida Zhou, Charles Fleming, Zhaoran Wang, Alfredo Garcia, and Mingyi Hong. HiPER: Hierarchical reinforcement learning with explicit credit assignment for large language model agents, 2026. URL https://arxiv.org/abs/2602.16165v2.

Hongjin Qian, Zhao Cao, and Zheng Liu. MemoBrain: Executive memory as an agentic brain for reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 2646–2662, San Diego, California, United States, July 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.127. URL https://aclanthology.org/2026.findings-acl.127/.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

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

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for LLM reasoning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/98711dea460bdefe0e651ca23ec98ba2-Abstract-Conference.html. Spotlight.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.775. URL https://aclanthology.org/2022.emnlp-main.775/.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. ReSum: Unlocking long-horizon search intelligence via context summarization, 2026. URL https://arxiv.org/abs/2509.13313v3. Version 3, revised March 2026.

Yaxiong Wu, Sheng Liang, Chen Zhang, Yichao Wang, Yongyue Zhang, Huifeng Guo, Ruiming Tang, and Yong Liu. From human memory to AI memory: A survey on memory mechanisms in the era of LLMs, 2025. URL https://arxiv.org/abs/2504.15965v2. Version 2, April 23, 2025.

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts LLM agents: An empirical study of experience-following behavior. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 623–645, San Diego, California, United States, July 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.27. URL https://aclanthology.org/2026.acl-long.27/.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, pp. 17577–17604. Curran Associates, Inc., 2025. doi: 10.52202/085713-0593. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schuetze, Volker Tresp, and Yunpu Ma. Memory-R1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.583. URL https://aclanthology.org/2026.acl-long.583/.

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

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21457–21483, San Diego, California, United States, July 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.981. URL https://aclanthology.org/2026.acl-long.981/.

Kehao Zhang, Shangtong Gui, Sheng Yang, Wei Chen, and Yang Feng. Learning to remember: End-to-end training of memory agents for long-context reasoning. *arXiv preprint arXiv:2602.18493*, 2026. URL https://arxiv.org/abs/2602.18493v2.

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. doi: 10.1609/aaai.v38i17.29946. URL https://ojs.aaai.org/index.php/AAAI/article/view/29946.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents, 2025. URL https://arxiv.org/abs/2506.15841v2.

# A METHOD AND TOOL SPECIFICATIONS

VerMem separates the information retained in memory from the information currently available to the task solver. The tool contracts specify how commands change these states and when a transition is admissible. Their semantic predicates are distinct from the graded feedback supplied by the training verifiers.

## A.1 RUNTIME STATE AND EXECUTION PROTOCOL

VerMem represents the memory-decision state as $s_t = (q, M_t, C_t, H_t)$. The task specification $q$ remains fixed within one task instance. The long-term memory $M_t$ contains persistent entries that can be reused in later steps or stages. Each entry records its content, source, current status, and revision history. The active context $C_t$ contains the budgeted information currently visible to the task solver. The episodic history $H_t$ preserves the ordered record of observations, task actions, tool outputs, intermediate conclusions, and memory-related events from the same task. The independent rollout record stores every memory decision, including null; it is not identical to $H_t$.

The active context and episodic history serve different purposes. Removing information from $C_t$ reduces the content visible to the task solver, but does not remove the corresponding event from $H_t$. This separation allows earlier evidence to be recovered after the task focus changes.

At each memory-decision step, the policy generates a complete command $u_t = (v_t, \xi_t)$, where $v_t \in \mathcal{V} \cup \{\varnothing\}$. Each non-null command specifies one atomic tool and its structured arguments. The policy samples $u_t$ from the bounded serialization $x_t = g(s_t)$ defined in Equation (1). The policy receives a bounded serialization of the task specification and budgeted views of $M_t$, $C_t$, and $H_t$. This policy-visible serialization is distinct from the active-context budget governing $C_t$. Before the command changes the real memory state, the proposed transition is checked against the tool schema, target state, required arguments, source information, and current budget. The transition follows Equation (2). Each valid non-null command commits its realized memory transition. The null command is the immediate identity $\widetilde{s}_t = s_t$, including $H_t$. Its decision is stored in the independent rollout record; subsequent task actions and observations belong to $\mathcal{T}_e$. If a non-null command is rejected, $M_t$ and $C_t$ remain unchanged, while the failed command and its reason are recorded in $H_t$. During training, the corresponding violation is recorded through $c_{t,k}$.

After the memory transition, the task solver continues with the resulting active context. The task action and resulting observation are then incorporated through the environment interaction transition in Equation (3), which constructs $s_{t+1}$. Figure A.2 distinguishes the execution branches. The following procedure summarizes this runtime process. The local and global verifiers are used only during training and do not participate in the reported inference process.

**Unified runtime procedure.** Starting from $M_0$, $C_0$, and $H_0$, each memory-decision step performs the following operations:

1. construct $s_t = (q, M_t, C_t, H_t)$ and its bounded serialization $x_t = g(s_t)$;
2. sample the complete command $u_t \sim \pi_\theta(\cdot \mid x_t)$;
3. compute $\widetilde{s}_t = \mathcal{T}_m(s_t, u_t)$ under the applicable execution branch. The valid non-null command commits its realized transition. The valid null command preserves the entire state, $\widetilde{s}_t = s_t$, and is recorded independently of $H_t$. The rejected non-null command preserves $M_t$ and $C_t$, records the failed command and reason in $H_t$, and receives the corresponding $c_{t,k}$ during training;
4. sample the task action $b_t$ from the resulting active context, collect the next observation, and construct $s_{t+1} = \mathcal{T}_e(\widetilde{s}_t, b_t, z_{t+1})$;
5. repeat until task termination or a configured rollout limit.

## A.2 ATOMIC TOOL SCHEMAS AND STATE TRANSITIONS

Figures A.1 and A.2 illustrate the tool contracts and execution branches, rather than sampled rollout records.

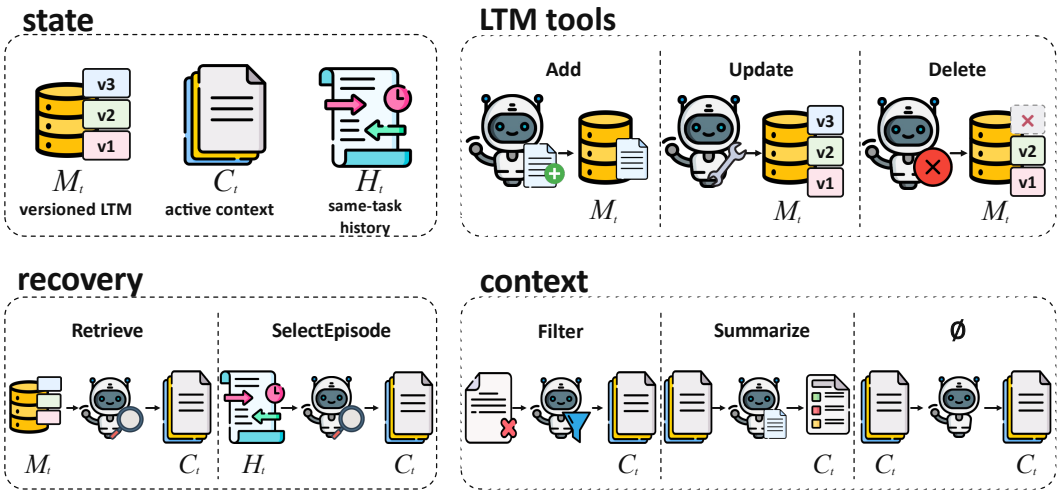



Figure A.1: Memory states and tool effects, including versioned updates, historical recovery, and the null operation.

Each non-null memory command selects one of the seven tools and supplies its required arguments. Table 6 specifies the required information, state transition, and principal validity condition of each tool.

Table 5: Seven non-null memory tools. The additional null command gives $\widetilde{s}_t = s_t$ under the immediate memory transition. Retrieve and SelectEpisode have different sources.

| Tool | Target | State effect |
|---|---|---|
| Add | LTM | Add a source-bearing entry to $M_t$. |
| Update | LTM | Create a revised entry version; retain its predecessor. |
| Delete | LTM | Soft-delete an entry; retain its record. |
| Retrieve | Context | Introduce active LTM entries: $M_t \to C_t$. |
| Filter | Context | Remove selected content from $C_t$. |
| SelectEpisode | Context | Restore same-task fragments: $H_t \to C_t$. |
| Summarize | Context | Replace selected content with a faithful shorter form. |

**Add.** Add stores new information with persistent utility. The new entry records the information and its source. The operation is rejected when the content is empty, unsupported, or duplicates an existing active entry. When an existing entry requires revision, Update is used instead.

**Update.** Update revises an existing LTM entry. It creates a new version rather than destructively overwriting the previous content. This retains the relation between the earlier and current versions and allows later inspection of how the memory state changed.

**Delete.** Delete follows soft-deletion semantics. The target entry is excluded from subsequent ordinary retrieval, while its content and deletion record remain available for inspection.

**Retrieve.** Retrieve is categorized as an STM tool because it does not change $M_t$. Instead, it selects active LTM entries and introduces them into $C_t$. The retrieval query is generated as part of $\xi_t$. Each operation returns at most $1{,}024$ tokens from active LTM. The resulting $C_t$ must also fit its $8{,}192$-token capacity before the content is committed. Deleted entries cannot be retrieved.

**Filter.** Filter removes selected low-value content from $C_t$. Task instructions, the current environment state, and indispensable evidence are protected. Filtering changes the active context but does not remove the original event from $H_t$.

**SelectEpisode.** SelectEpisode performs episode-level context selection over the current task history. It identifies earlier observations, tool outputs, or intermediate conclusions that are relevant to

Table 6: Compact specifications of the atomic memory tools in VerMem. The table reports the information required by each operation, its state transition, and its principal validity condition.

| **Tool and Required Information** | **State Transition** | **Key Validity Condition** |
|---|---|---|
| Add: new content and its source | Create a new persistent entry in $M_t$. | The content is non-empty, supported, and not a duplicate of an active entry. |
| Update: target entry, revised content, and source | Create a revised version in $M_t$ while retaining the previous version. | The target exists, and the revision is supported by the current evidence. |
| Delete: target entry and deletion reason | Mark the target entry as deleted without removing its history. | The target exists, and its deletion is justified by the current state. |
| Retrieve: query | Introduce relevant active entries from $M_t$ into $C_t$. | Deleted entries are excluded. The operation follows the configured active-context budget. |
| Filter: context items to remove | Remove the selected low-value items from $C_t$. | The selected items belong to $C_t$, and indispensable evidence is preserved. |
| SelectEpisode: historical fragments to restore | Introduce selected same-task fragments from $H_t$ into $C_t$. | The fragments belong to the current task history and fit the active-context budget. |
| Summarize: context items and their summary | Replace selected content in $C_t$ with a shorter summary. | The summary is faithful to its sources and preserves task-relevant information. |

the current task state and returns them to $C_t$. The operation cannot access history from another task instance. Each operation restores at most $1{,}024$ tokens, and the resulting $C_t$ must fit its $8{,}192$-token capacity.

**Summarize.** Summarize replaces selected content in $C_t$ with a shorter representation. The summary must remain faithful to the source content and preserve information required by the current task. The complete Summarize command, including the summary, source references, and other arguments, contains at most $512$ output tokens. The resulting $C_t$ must fit its $8{,}192$-token capacity. The $1{,}024$-token return limit for Retrieve and SelectEpisode does not apply to Summarize. The original events remain available in $H_t$.

### A.3 Historical-Context Recovery via Episode-Level Context Selection

Historical-context recovery concerns information that has left the active context but becomes useful again at a later task stage. It operates over the episodic history of the current task. It is therefore distinct from retrieving persistent information from LTM.

The system first identifies same-task historical fragments that are relevant to the current task specification and active context. The memory operation policy then selects the fragments to be restored. Selected content is restored to $C_t$ subject to the configured active-context budget and retains its original source and temporal position in $H_t$.

The two information paths are

$$M_t \xrightarrow{\text{Retrieve}} C_t, \qquad H_t \xrightarrow{\text{SelectEpisode}} C_t.$$

Although both operations introduce information into the active context, their information sources and intended uses are different. Table 7 summarizes this distinction.

Figure A.1 distinguishes these sources. They let the policy recover earlier task evidence without requiring that it first be written to persistent memory.

Table 7: Difference between LTM retrieval and historical-context recovery through episode-level context selection.

| Property | Retrieve | SelectEpisode |
|---|---|---|
| Source | Long-term memory $M_t$ | Episodic history $H_t$ |
| Content | Persistent facts and reusable knowledge | Earlier observations, tool outputs, and intermediate conclusions |
| Scope | The current LTM state | The current task history |
| Motivation | Stored knowledge is absent from $C_t$ | Earlier task evidence becomes relevant again |
| Target | $C_t$ | $C_t$ |
| Main risk | Irrelevant or outdated persistent information | Redundant or misleading historical fragments |

### A.4 State Validation and Failure Handling

The executor checks each non-null command against its tool schema, references, preconditions, and active-context budget before committing its state transition. The tool contracts in Appendix A.2 require source-supported writes, justified deletion, and preservation of task-essential evidence. Protected content comprises the task instruction, current environment state, and evidence indispensable to continued solving. These requirements concern semantic validity as well as field completeness. Retaining a source in $H_t$ and recovering it within the remaining decision and token budgets are distinct conditions. The commit and rejection branches follow Appendix A.1; their training costs are listed in Table 18.

Figure A.2 separates valid non-null, valid null, and rejected commands.

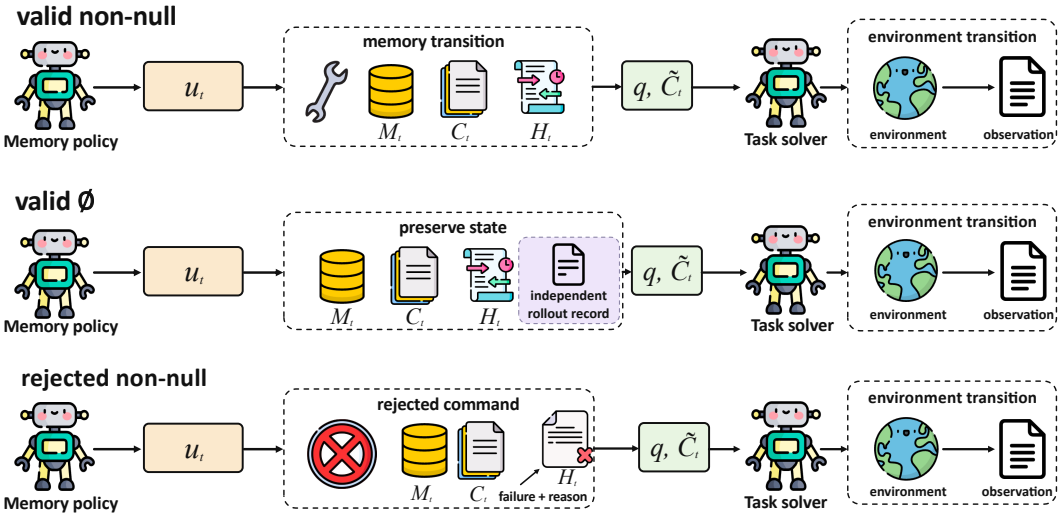



Figure A.2: Execution branches for valid non-null, valid null, and rejected commands.

If a proposed command is invalid, $M_t$ and $C_t$ remain unchanged. The failed command and its reason are retained in the task history. During training, the corresponding structural, safety, or budget violation contributes to $c_{t,k}$. The task solver then continues with the unchanged active context unless the task itself has reached a terminal failure.

Update must preserve the relation between memory versions. Delete must remain soft. Filter must not remove indispensable evidence. SelectEpisode must remain within the current task history. Summarize must reduce context length without changing the supported meaning. Retrieve, SelectEpisode, and Summarize must also satisfy the active-context budget before their transitions are committed.

Execution checks determine command admissibility and are separate from the training verifiers. The local verifier evaluates the task relevance, evidence grounding, local progress, and information fidelity of a realized operation. The global verifier supplies only evidence-coherence and terminal-memory-consistency scores after trajectory completion. These scores enter the composite global

branch together with programmatic task, supporting-fact-recall, and efficiency components. Both verifiers are disabled during inference.

# B Optimization Objective and Loss Scope

The optimization unit is a complete memory command, including its structured arguments. The settings below define its loss scope, the candidate groups used for learning, and the separate scalars used to monitor training. Data construction and scoring are specified in Appendices D and F.

## B.1 Core Training Configuration and Loss Scope

Table 8 gives the shared training settings. Its token limits constrain different objects: policy-state serialization, active context, generated responses, and tool returns.

The limits apply to the two Qwen backbones in VerMem's training and internal evaluation configurations, for the roles and tools enabled in each configuration. Phase A retains its LTM-only operation subspace; Retrieve, SelectEpisode, and Summarize are enabled in Phases B and C. The active context $C_t$ and policy-state serialization $g(s_t)$ each have a capacity of $8{,}192$ tokens and are checked separately. The former contains the task context and information currently supplied to the solver; the latter is the policy's bounded state input. Neither object is automatically identical to the corresponding complete model prompt. The solver and command limits constrain generated outputs, whereas online TN counts actual request inputs and outputs (Appendix C.7).

During reinforcement learning, all generated tokens belonging to one atomic memory operation share the same $A^{\text{hier}}_{t,k}$. Different memory decisions can receive different advantages. The policy objective is applied only to tokens that serialize the complete command $u_{t,k}$, including its structured arguments. Task-solver outputs, environment observations, input-state tokens, and padding tokens do not receive memory-operation credit. Phase A, Phase B, and Phase C save separate checkpoints. Each phase initializes the next one through policy parameters only.

## B.2 Complete Command and Monitoring Definitions

The operation sets are

$$\begin{gathered}\mathcal{V}_{\text{LTM}} = \{\textsc{Add}, \textsc{Update}, \textsc{Delete}\},\\ \mathcal{V}_{\text{STM}} = \{\textsc{Retrieve}, \textsc{Filter}, \textsc{SelectEpisode}, \textsc{Summarize}\},\\ \mathcal{V} = \mathcal{V}_{\text{LTM}} \cup \mathcal{V}_{\text{STM}}.\end{gathered} \tag{11}$$

The SFT command-target objective is

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{(x_i, v_i^\star, \xi_i^\star)\sim\mathcal{D}_{\text{SFT}}}\left[\log \pi_\theta(v_i^\star, \xi_i^\star \mid x_i)\right]. \tag{12}$$

The dataset $\mathcal{D}_{\text{SFT}}$ contains bounded states and validated command targets. The stateful candidate group for phase $p$ and update $n$ is

$$\begin{gathered}\tau_{k,p}^{(q,n)} \sim \mathbb{P}_{p,n}\left(\tau \mid s_{0,p}^{(q)}\right), \quad k = 1, \ldots, K,\\ \mathcal{G}_{q,p}^{(n)} = \{\tau_{1,p}^{(q,n)}, \ldots, \tau_{K,p}^{(q,n)}\}.\end{gathered} \tag{13}$$

All candidates for the task share the initial state $s_{0,p}^{(q)}$. Equation (18) defines the token ratios and KL terms in the command objective, with separate update-wise old policies and phase-wise reference policies. Let $T_k$ count all memory decisions, including null and rejected commands. The executable set $\mathcal{E}_k$ includes valid null decisions scored by the null rubric. The monitor components are

$$R^{\text{local}}(\tau_k) = \frac{1}{T_k}\sum_{t\in\mathcal{E}_k} r_{t,k}^{\text{local}}. \tag{14}$$

$$P_{\text{aux}}(\tau_k) = \frac{1}{T_k}\sum_{t=1}^{T_k} c_{t,k}. \tag{15}$$

Table 8: Core training and rollout configuration used by VerMem. Token limits are measured in model tokens.

| Setting | Value | Setting | Value |
|---|---|---|---|
| SFT train/dev size | 32,000 / 4,000 | Task batch per update | 8 |
| SFT epochs | 3 | Candidate-group size $K$ | 8 |
| SFT learning rate | $2 \times 10^{-5}$ | Phase A/B/C updates | 1,000 / 1,000 / 1,500 |
| SFT effective batch | 64 | Rollout temperature | 0.7 |
| SFT warmup ratio | 0.03 | Rollout top-$p$ | 0.95 |
| Maximum gradient norm | 1.0 | GRPO clip range | 0.2 |
| RL learning rate | $1 \times 10^{-6}$ | KL coefficient | 0.1 |
| Policy-state serialization limit | 8,192 tokens | Task-solver response limit | 2,048 output tokens per call |
| Memory-command output limit | 512 tokens per complete command | Active-context capacity | 8,192 tokens |
| Retrieve return limit | 1,024 tokens per operation | SelectEpisode return limit | 1,024 tokens per operation |
| Random seeds | 42, 43, 44 | Valid null accounting | Horizon: yes; tool calls: no |

The local sum is zero when no command is executable. The strategy-specific monitor mappings are

$$m_k^{\text{all}} = \text{clip}\left(\frac{R^{\text{local}}(\tau_k) + r_k^{\text{global}} - P_{\text{aux}}(\tau_k)}{2}, 0, 1\right). \tag{16}$$

$$m_k^{\text{ans}} = \text{clip}\big(r_k^{\text{task}} - P_{\text{aux}}(\tau_k), 0, 1\big). \tag{17}$$

These scalars are used for monitoring, not in place of the normalized command advantages. Their absolute levels are not directly comparable across strategies.

### B.3 Input Permissions by Model Role

Table 9: Information available to each model or scoring role. Reference answers and supporting-fact labels are confined to construction, reward, and evaluation paths. Runtime execution checks are distinct from these training verifiers.

| Role | Visible input | Labels and use |
|---|---|---|
| Memory policy | Bounded $g(q, M, C, H)$ | No reference labels; training and evaluation. |
| Task solver | $q, \widetilde{C}_t$ | No reference labels; task execution. |
| SFT teacher | Bounded state, allowed tools, visible evidence | No reference answer or future observations. |
| Local verifier | Bounded state, command, realized transition, visible sources | No final answer or later states; training only. |
| Global verifier | Completed trajectory, terminal memory, final answer | Reference answer and supporting facts; training only. |
| Programmatic reward | Answer, evidence representation, resource counts | Reference labels for task and recall scores. |
| Qwen-Max judges | Question and answer; active terminal LTM for MQ | Reference answer; supporting facts for MQ; held-out evaluation. |

Table 9 summarizes model-role inputs and label permissions. The answer judge receives the question, reference answer, and agent answer; the MQ judge receives the question, reference answer, supporting facts, and active terminal LTM. Their original prompts appear in Appendix C. Runtime execution checks remain distinct from these model roles.

## C Experimental Configuration and Evaluation Protocol

The VerMem comparisons evaluate task performance, memory quality, and resource use under the protocols below. Main-table Base and external baseline values are reproduced from Yu et al. (2026)

and retain their published execution settings. Budget-controlled evaluation and online-token accounting have separate definitions in Appendix C.7; the training data and reward signals are specified in Appendices D and F.

### C.1 Benchmarks and Evaluation Splits

We evaluate VerMem on ALFWorld, SciWorld, PDDL, BabyAI, and HotpotQA. ALFWorld, SciWorld, PDDL, and BabyAI use their official test splits. HotpotQA uses the official distractor validation split as the held-out evaluation set because it provides both reference answers and annotated supporting facts. The HotpotQA training split is used only for the SFT and reinforcement-learning instances described in Appendix D. Original question identifiers do not overlap between the construction-training pool, the internal development pool, and the held-out evaluation set.

ALFWorld, SciWorld, and BabyAI use environment success as the task outcome. PDDL uses the progress value returned by the benchmark. One HotpotQA question defines one evaluation episode. The question remains fixed as $q$, while the distractor passages enter the interaction as task observations. Supporting-fact annotations are not exposed to the memory operation policy. They are used for training-instance construction, programmatic supporting-fact recall, verifier inputs, and evaluation. The reference answer is likewise hidden from the memory operation policy.

Table 10 summarizes the evaluation split and reported metric for each benchmark. Every executable instance in the selected split is evaluated. The VerMem variants use the same task order, environment initialization, and information source.

Table 10: Evaluation splits and primary metrics. Supporting-fact annotations are not exposed to the memory operation policy. They are used for training-instance construction, programmatic supporting-fact recall, verifier inputs, and evaluation.

| **Benchmark** | **Evaluation Split** | **Reported Metric** | **Evaluation Focus** |
|---|---|---|---|
| ALFWorld | Official test split | Success Rate | Long-horizon embodied interaction and environment-state tracking. |
| SciWorld | Official test split | Success Rate | Scientific procedure execution and maintenance of intermediate observations. |
| PDDL | Official AgentBoard test split | Progress Rate | Symbolic planning progress and intermediate goal-state maintenance. |
| BabyAI | Official test split | Success Rate | Instruction following, navigation, and short-term state tracking. |
| HotpotQA | Official distractor validation split | LLM-as-a-Judge $J_{\text{judge}}$ | Multi-hop evidence aggregation, persistent-memory maintenance, and active-context control. |

### C.2 State Isolation and Episode Initialization

Every task instance uses an isolated memory state. Each episode starts with empty $M_0$ and $H_0$, while $C_0$ contains only the task instruction and the initial input. The initial input for ALFWorld, SciWorld, PDDL, and BabyAI is the first observation returned by the benchmark. For HotpotQA, it contains the question and the passage content currently exposed by the task environment.

The memory states persist only within one episode. After task completion, failure, or budget termination, $M_t$, $C_t$, and $H_t$ are released. This reset applies to the VerMem evaluation. The cited external main-table scores retain the provenance of their published evaluation.

### C.3 Backbones and Inference Configuration

The main experiments use Qwen2.5-7B-Instruct and Qwen3-4B-Instruct. VerMem and VerMem-noVerify use the corresponding backbone. External main-table values use the backbone settings reported by Yu et al. (2026).

Evaluation uses deterministic decoding. The memory operation policy and task solver use temperature 0 and top-$p$ 1.0. VerMem constructs $x_t = g(s_t)$ under the 8,192-token policy-state serialization limit. The active context $C_t$ has a separate 8,192-token capacity. The task solver generates at most 2,048 output tokens per call, and each complete memory command contains at most 512 output tokens. Retrieve and SelectEpisode each return at most 1,024 tokens per operation. Both return limits and the operation's resulting active-context capacity must be satisfied. Summarize follows the complete-command and active-context constraints in Appendix A.2. The valid null command does not count as a memory-tool call or in TC, but it remains a memory decision and counts toward the memory-decision horizon.

The local and global verifiers are disabled during evaluation. Table 11 collects these inference settings.

### C.4 Baseline Configurations

The Base, LangMem, A-Mem, Mem0, Mem0$^g$, and AgeMem entries in Table 1 are reproduced from Yu et al. (2026). LangMem provides modular storage, search, and consolidation; A-Mem uses dynamic linking and evolution; Mem0 and Mem0$^g$ use conversational fact and graph workflows. AgeMem learns unified LTM/STM control. Their published numbers are reference comparisons and do not establish matched executions in the present implementation.

VerMem-noVerify and VerMem share the architecture, backbone, complete command space, SFT initialization, Phase A/B/C curriculum, GRPO settings, and transition constraints. VerMem-noVerify uses the normalized task-outcome advantage and constraints, without the local operation advantage or composite global trajectory advantage. Full VerMem uses both branches. This internal control tests the full feedback configuration rather than the semantic global verifier alone.

### C.5 Metric Computation

Success Rate on ALFWorld, SciWorld, and BabyAI is the number of successful episodes divided by the number of evaluated episodes. PDDL uses the normalized Progress Rate returned by the benchmark and reports it on a 0–100 scale. HotpotQA uses the Qwen-Max LLM-as-a-Judge score $J_{\text{judge}}$, and the main table reports $100J_{\text{judge}}$.

The Average column is the arithmetic mean of ALFWorld SR, SciWorld SR, PDDL PR, BabyAI SR, and HotpotQA $100J_{\text{judge}}$. Every benchmark therefore receives equal weight.

The reward-function ablation additionally reports TN, MQ, and TC. TN is the mean number of online input and output tokens from online model calls in one HotpotQA episode, including the memory policy and task solver. Repeated inputs and generated null or rejected commands contribute to TN. MQ evaluates the active entries in terminal $M_t$ against the annotated supporting facts. Soft-deleted entries are excluded. TC is the mean number of non-null memory-tool calls. The null action $\varnothing$ is not counted in TC, but it remains part of the memory-decision horizon.

### C.6 Qwen-Max Evaluation

The HotpotQA $J_{\text{judge}}$ and MQ metrics use the fixed `qwen-max-2025-01-25` snapshot. The evaluator is independent of the DeepSeek-V3.2 teacher and the local and global verifiers. It does not generate SFT targets, provide training rewards, or participate in a memory transition.

Qwen-Max evaluates each example independently with temperature 0, top-$p$ 1.0, and at most 16 output tokens. Its output contains one number in $[0, 1]$. Each judge permits one format retry with an appended format reminder. Appendix F.1 distinguishes scoring responses from command-execution outcomes.

The answer judge compares the question, ground-truth answer, and agent answer. Semantically equivalent concise answers can receive full credit. Partially correct answers receive an intermediate score, while irrelevant or contradictory answers receive a score close to zero. The following block gives the complete answer-judge prompt. Bracketed fields in this and subsequent prompt blocks denote values inserted at runtime.

Table 11: VerMem inference configuration. Token limits are measured with the tokenizer of the corresponding backbone.

| Setting | Value | Setting | Value |
|---|---|---|---|
| Policy temperature | 0 | Policy top-$p$ | 1.0 |
| Task-solver temperature | 0 | Task-solver top-$p$ | 1.0 |
| Policy-state serialization limit | 8,192 tokens | Task-solver response limit | 2,048 output tokens per call |
| Memory-command output limit | 512 tokens per complete command | Active-context capacity | 8,192 tokens |
| Retrieve return limit | 1,024 tokens per operation | SelectEpisode return limit | 1,024 tokens per operation |
| Evaluation seeds | 42, 43, 44 | Local/global verifiers | Disabled |
| Memory reset | After every episode | Null-decision accounting | Horizon: yes; TC: no |

**Qwen-Max answer-judge prompt.**

```
You are an independent judge evaluating the correctness
of an answer to a question.

Question:
[QUESTION]

Ground-truth answer:
[GROUND_TRUTH_ANSWER]

Agent answer:
[AGENT_ANSWER]

Score the agent answer from 0.0 to 1.0.

1.0: fully correct or semantically equivalent
0.8-0.9: correct with only a minor omission or wording issue
0.6-0.7: partially correct but missing an important element
0.4-0.5: contains some correct information and a major error
0.2-0.3: mostly incorrect with limited relevant content
0.0-0.1: incorrect, contradictory, or irrelevant

Return only one number between 0.0 and 1.0.
Do not provide an explanation.
```

The MQ evaluator receives the question, reference answer, annotated supporting facts, and active terminal LTM entries. It evaluates supporting-fact coverage, relevance to the question, and irrelevant stored content. The following block gives the complete MQ prompt.

**Qwen-Max memory-quality prompt.**

```
You are an independent judge evaluating the quality of
long-term memory for question answering.

Question:
[QUESTION]

Reference answer:
[REFERENCE_ANSWER]

Annotated supporting facts:
[SUPPORTING_FACTS]

Active entries in the terminal long-term memory:
[TERMINAL_LTM]

Evaluate the terminal long-term memory on three criteria:
```

```
1. Does it cover the supporting facts needed to answer the
   question?
2. Are the stored entries relevant and factually consistent?
3. Is it free of irrelevant, duplicated, or misleading
   content?

Score the memory from 0.0 to 1.0.

1.0: complete, relevant, consistent, and free of noise
0.8-0.9: nearly complete with only a minor omission or
         irrelevant entry
0.6-0.7: useful but missing important supporting content
0.4-0.5: contains some useful evidence and substantial noise
0.2-0.3: limited relevant content with major errors
0.0-0.1: unsupported, irrelevant, or unusable memory

Return only one number between 0.0 and 1.0.
Do not provide an explanation.
```

MQ is used only for the reward-function analysis and is not included in the Average column of the main results.

### C.7 Budget-Controlled Efficiency Evaluation and Token Accounting

Figure 4 compares A-Mem, Mem0, AgeMem, and VerMem with Qwen2.5-7B-Instruct on ALFWorld, SciWorld, and BabyAI. Success Rate is computed separately for the three benchmarks and then macro-averaged with equal weight. The curves show the mean over seeds $42$, $43$, and $44$; shaded regions denote one standard deviation across the three runs. These budget-controlled comparisons are distinct from the cited external rows of Table 1.

The online token budgets range from $1{,}000$ to $5{,}000$ tokens. Each budget is enforced as a hard episode limit; an unfinished episode is counted as unsuccessful when its budget is exhausted. The targets $S^* = 40.0$ and $B^* = 2{,}500$ are selected from validation curves and fixed before test evaluation. $\mathrm{SR@}B^*$ is the macro-average SR at budget $B^*$. $\mathrm{TN@}S^*$ denotes the interpolated threshold budget at which the mean success curve first reaches $S^*$, rather than the actual tokens consumed by an episode. Table 2 reports these two metrics.

Threshold computation uses the first upward crossing of the mean success curve. Linear interpolation between the two curve points bracketing $S^*$ gives $\mathrm{TN@}S^*$. Interpolation follows benchmark and seed aggregation; it is not an average of per-seed crossings. The fixed-budget score and interpolated threshold describe different properties of the same mean curve.

Online accounting counts every actual input and output token, including repeated inputs, null commands, and rejected commands. It includes the memory policy, task solver, any online generative memory module in a baseline, and any additional online model-based execution check. Training, offline embedding and index construction, training local/global verifier calls, and offline Qwen-Max evaluation are outside this online measure. The count includes system templates, separately supplied questions, and wrapper tokens whenever they enter an actual online request. Their membership in the logical objects $C_t$ or $g(s_t)$ depends on request assembly; the object capacities do not define a cap on the complete prompt or the episode's accumulated TN. In Table 4, TN is the mean actual online input and output consumption per HotpotQA episode, not the threshold-budget metric $\mathrm{TN@}S^*$.

### C.8 Random Seeds and Statistical Reporting

The training and evaluation protocol specifies seeds $42$, $43$, and $44$, with a separate SFT, Phase A/B/C, and downstream-evaluation sequence for each seed. Within each seed, the protocol pairs the VerMem variants on benchmark instances and task order.

The main-table VerMem values are reported as means over these seeds. Base and external baseline values retain the published evaluation protocol of Yu et al. (2026). Source and comparison scope for the cumulative configurations are given in Appendix E.

Phase C checkpoints are selected only with the internal development set. Official benchmark test results, held-out HotpotQA $J_{\text{judge}}$, and MQ are never used for checkpoint selection. Evaluator prompts and evaluation decoding settings are fixed before test evaluation.

# D Training Data and Stateful Rollout Construction

The HotpotQA construction pipeline produces supervised command targets and phase-specific stateful episodes. Original question identifiers separate the construction and development pools; teacher generation and quality filtering produce command targets, while independent initial states define the candidates used in each reinforcement-learning phase.

## D.1 HotpotQA Training Data and Split

All SFT and reinforcement learning instances are derived from the HotpotQA training split (Yang et al., 2018). Each original example provides a question $q$, a reference answer, annotated supporting facts, and distractor passages. The question remains fixed within one training episode. The reference answer supports instance construction and terminal task signals, but it is not exposed to the memory operation policy. Supporting-fact annotations are not exposed to the memory operation policy. They are used for training-instance construction, programmatic supporting-fact recall, verifier inputs, and evaluation.

We divide the original question identifiers in the HotpotQA training split into a $90\%$ construction-training pool and a $10\%$ internal-development pool. All states and episodes derived from one original question remain in the same partition. The official HotpotQA validation and test splits are not used to construct SFT data or Phase A, B, or C episodes.

Training construction uses supporting and distractor passages as source-bearing evidence units. Each unit retains its document title, sentence position, source identifier, original text, and original question and split membership. The partial, conflicting, duplicated, and obsolete states used for Update and Delete are constructed training conditions, rather than native HotpotQA annotations.

The final SFT training set contains $32{,}000$ states, and the internal development set contains $4{,}000$ states. Add, Update, Delete, Retrieve, Filter, SelectEpisode, Summarize, and the null action $\varnothing$ are balanced in both sets. Each decision contributes $4{,}000$ training examples and $500$ development examples. Half of each class is derived from high-precision boundary templates, and half is retained from teacher-generated candidates. At least $25\%$ of each class consists of boundary or negative decision states.

## D.2 SFT Data Generation and Teacher Prompt

Boundary templates construct memory-decision states with a well-defined target. They emphasize distinctions that are easy to confuse, including Add versus Update, Retrieve versus SelectEpisode, Filter versus Summarize, and an explicit operation versus $\varnothing$. For example, when a fact is already represented in $M_t$ and no new evidence is available, the target is $\varnothing$ rather than a duplicate Add. When the missing evidence appears only in $H_t$, the target is SelectEpisode rather than Retrieve.

The frozen DeepSeek-V3.2 teacher expands the diversity of operation choices and structured arguments. For each state, the teacher receives the bounded policy input $x_i$, the operation-type subspace allowed by the current scenario, and the source evidence already visible in the state. The reference answer and future observations are not provided.

For each state sent to the teacher, two candidates are sampled independently with temperature $0.2$, top-$p$ $0.95$, and at most $512$ output tokens per candidate. Each candidate contains one complete command. This sampling count does not determine the number of accepted targets in the final SFT dataset.

The teacher prompt requests exactly one complete atomic operation or $\varnothing$, with no accompanying analysis. Candidate acceptance and the retained SFT targets are defined separately below.

**SFT teacher prompt.**

```
You are a teacher model for constructing supervised
examples for VerMem.

Given a task and the current memory-decision state, select
exactly one atomic memory operation, or select the null
action when no memory change is needed.

Use only information that is already visible in the state.
Do not use the reference answer, future observations, or
unsupported knowledge.

Available operations:
1. Add
   Use only when new information has persistent value and
   is not already represented in long-term memory.

2. Update
   Use when new evidence revises an existing long-term
   memory entry.

3. Delete
   Use only when an existing entry is clearly incorrect,
   obsolete, duplicated, or unsupported.

4. Retrieve
   Use when required information exists in long-term memory
   but is missing from the active context.

5. Filter
   Remove irrelevant active-context items while preserving
   all evidence needed by the current task.

6. SelectEpisode
   Restore task-relevant fragments from the episodic history
   of the current task.

7. Summarize
   Replace overlong active-context content with a shorter
   representation that preserves critical evidence.

8. Null action
   Use when the current memory state requires no change.

Output exactly one of the following forms:

Add(content="...", source_refs=[...])

Update(memory_id="...", new_content="...",
       source_refs=[...])

Delete(memory_id="...", reason="...")

Retrieve(query="...")

Filter(drop_refs=[...])

SelectEpisode(selected_refs=[...])

Summarize(source_refs=[...], summary="...")

\varnothing

Do not provide analysis, explanations,
markdown, or more than one operation.
```

```
Task:
[TASK]

Long-term memory:
[LONG_TERM_MEMORY]

Active context:
[ACTIVE_CONTEXT]

Episodic history:
[EPISODIC_HISTORY]

Allowed operation-type subspace:
[ALLOWED_ACTIONS]

Visible source evidence:
[VISIBLE_EVIDENCE]
```

### D.3 SFT Quality Filtering and Optimization

Teacher outputs are not used directly as supervision labels. Candidate acceptance requires the following conditions:

1. the output contains exactly one operation;
2. the operation belongs to the permitted operation-type subspace;
3. all required arguments are present;
4. every referenced memory entry, context item, or historical fragment exists;
5. all references belong to the same HotpotQA training instance;
6. the realized transition satisfies the tool semantics and preconditions in Appendix A;
7. the output contains no answer leakage, future information, or unsupported content.

The first five conditions concern structure and reference scope; the last two also require semantic judgments about source support and permissible information. Candidates that fail the acceptance conditions are discarded without automatic repair. Valid candidates are then deduplicated according to their state, target operation, and structured arguments. When two examples define equivalent states and equivalent transitions, only one is retained.

Negative decision states never use an invalid operation as the supervision target. The generator changes information completeness, candidate location, distractor density, or memory consistency so that a superficially plausible operation becomes inappropriate. The target is another legal operation, corrected arguments, or $\varnothing$. Duplicate Add candidates may therefore be corrected to Update or $\varnothing$. Retrieve is replaced with SelectEpisode when the missing evidence appears only in $H_t$.

Table 12 reports the final SFT coverage. Each target contributes $4{,}000$ training examples and $500$ development examples.

Table 12: Balanced target-command coverage in the SFT training and internal development sets.

| Target Decision | Training | Development |
|---|---|---|
| Add | 4,000 | 500 |
| Update | 4,000 | 500 |
| Delete | 4,000 | 500 |
| Retrieve | 4,000 | 500 |
| Filter | 4,000 | 500 |
| SelectEpisode | 4,000 | 500 |
| Summarize | 4,000 | 500 |
| $\varnothing$ | 4,000 | 500 |
| **Total** | **32,000** | **4,000** |

SFT uses AdamW for three epochs with a learning rate of $2 \times 10^{-5}$, a warmup ratio of $0.03$, a maximum gradient norm of $1.0$, and an effective batch size of $64$. Policy-state inputs and complete

target commands are capped independently at $8{,}192$ and $512$ tokens. The target-length filter applies to the complete SFT command; it is separate from the teacher's $512$-token generation cap.

The loss applies only to tokens that serialize the complete target command $u_i^\star = (v_i^\star, \xi_i^\star)$. Task descriptions, input states, system instructions, and padding tokens are excluded from the target. Checkpoint selection uses tool accuracy and executable-call rate on the development set. When two checkpoints have the same tool accuracy, the one with the lower invalid-operation rate is selected.

### D.4 Phase-Specific Episode Construction

Reinforcement learning follows the parameter curriculum in Equation (10). Each phase initializes its task states independently and transfers only the policy parameters from the previous phase. Table 13 summarizes the data size, operation-type subspace, and episode construction of each phase.

**Phase A.** Phase A contains $8{,}000$ training episodes and $800$ development episodes. The initial $M_0$ contains the phase-specific persistent state, $C_0$ contains the current observation, and $H_0$ begins with the current episode. Add, Update, Delete, and $\varnothing$ scenario families are sampled uniformly. The question $q$ is fixed and visible through $x_t = g(s_t)$ throughout the episode. Evidence text is revealed incrementally, and solving the known question is delayed until the end; reference answers, supporting-fact labels, and future observations remain hidden from the policy. The rollout terminates when solving is complete, the token budget is exhausted, or another rollout limit is reached. The task solver is conditioned on $q, \widetilde{C}_t$, as in Equation (3). Phase A and the +LT configuration restrict the memory policy to LTM maintenance and null; the policy-state serialization and solver context retain their separate capacity limits. This operation subspace governs memory commands, while the solver continues from the post-memory active context.

**Phase B.** Phase B contains $8{,}000$ training episodes and $800$ development episodes. Each instance uses supporting documents and distractor documents from the HotpotQA distractor setting. The initial $M_0$ contains phase-specific entries. The initial $C_0$ occupies part of the configured active-context budget, and $H_0$ contains same-task history. Retrieve, Filter, SelectEpisode, Summarize, and $\varnothing$ scenario families are sampled uniformly.

Historical-context recovery cases place required evidence in earlier same-task history while recent history contains related distractors. Retrieve accesses $M_t$, whereas SelectEpisode accesses $H_t$. Returned content must satisfy the configured active-context budget. Valid null and rejected commands count toward the memory-decision horizon; valid null commands do not count as memory-tool calls.

**Phase C.** Phase C contains $12{,}000$ training episodes and $1{,}200$ development episodes. Every episode requires at least one LTM operation and one STM operation. Some episodes require historical-context recovery so that SelectEpisode receives supervision in complete tasks. Documents, task feedback, and observations are revealed sequentially. Early Add or Update decisions may affect a later Retrieve, while Filter or Summarize may change the evidence available to subsequent reasoning. Valid null and rejected commands count toward the memory-decision horizon.

All phases use the independent $8{,}192$-token capacities for $g(s_t)$ and $C_t$. Actual solver calls generate at most $2{,}048$ output tokens, and complete memory commands contain at most $512$ output tokens. These limits do not enable tools outside the phase-specific operation subspaces in Table 13.

### D.5 Stateful Candidate-Trajectory Collection

At update $n$ of phase $p \in \{A, B, C\}$, the current memory operation policy samples eight tasks. For task $q$, the generator constructs the phase-specific initial state $s_{0,p}^{(q)}$ and copies it into $K = 8$ isolated rollouts according to Equation (13). All trajectories in one candidate group share the task, initial $M_0$, $C_0$, $H_0$, and environment condition. They differ only in policy sampling and the resulting operation sequence. Rollouts use temperature $0.7$ and top-$p$ $0.95$.

At every step, the memory operation policy generates one complete command $u_{t,k}$, including $\varnothing$. The valid non-null command commits its transition through $\mathcal{T}_m$. The null command is the immediate identity $\widetilde{s}_t = s_t$, including $H_t$. Its decision is stored in the independent rollout record; subsequent

Table 13: Phase-specific episode construction. Only policy parameters are transferred between phases; memory states are initialized independently.

| Phase | Train/Dev | Operation-Type Subspace | Episode Construction |
|---|---|---|---|
| Phase A | 8,000 / 800 | $\mathcal{V}_{\mathrm{LTM}} \cup \{\varnothing\}$ | Incremental facts, conflicting evidence, corrections, duplicates, obsolete entries, and delayed solving of the known question. |
| Phase B | 8,000 / 800 | $\mathcal{V}_{\mathrm{STM}} \cup \{\varnothing\}$ | Prepared LTM, distractor-rich active context, same-task episodic history, summarization, and historical-context recovery. |
| Phase C | 12,000 / 1,200 | $\mathcal{V} \cup \{\varnothing\}$ | Complete tasks with alternating LTM and STM operations; some tasks additionally require historical-context recovery. |

task actions and observations belong to $\mathcal{T}_e$. If a non-null command violates a precondition in Appendix A, $M_t$ and $C_t$ remain unchanged, the failure record is added to $H_t$, and the corresponding $c_{t,k}$ is recorded. The task solver then continues with the resulting active context. Each rollout record retains the bounded policy input $x_{t,k}$, the complete command $u_{t,k}$, the post-memory state $\widetilde{s}_t$, the post-interaction state $s_{t+1}$, and the command-token log probabilities. This preserves the transition distinction made in Equations (2) and (3).

Each trajectory terminates when the task solver produces a final answer, the environment terminates, the configured rollout limit is reached, or the token budget is exhausted. Budget-terminated trajectories remain in the candidate group and are processed by the composite global branch and constraint channel. The global verifier contributes only its evidence-coherence and terminal-memory-consistency components.

The completed group $\mathcal{G}_{q,p}^{(n)}$ supplies the task-conditioned reference for global credit. Executable commands from all groups in the update supply the operation-specific local references.

Phase A and Phase B each use $1{,}000$ policy updates, while Phase C uses $1{,}500$ updates. With eight tasks and eight trajectories per task, the schedule specifies $64{,}000$, $64{,}000$, and $96{,}000$ candidate trajectories for Phases A, B, and C, respectively.

## E Cumulative Configuration Comparison

Figure 5 in the main paper and Table 14 describe the sequence Base, +LT, +LT/ST, +noV, and +V under Qwen2.5-7B-Instruct. The configurations differ in their tool space and training. The +LT and +LT/ST configurations use SFT: +LT enables LTM operations, and +LT/ST enables the complete LTM/STM command space. The +noV and +V configurations use RL. The former uses normalized task-outcome advantage and hard constraints; the latter adds local and composite-global credit. Base provides the published reference scores from Yu et al. (2026), also listed in Table 1. The comparison places this reference alongside the four memory and training configurations. Its endpoints describe configuration-level performance; the intermediate settings distinguish tool coverage and feedback design rather than independent effects of single components.

Table 14: Task scores across cumulative configurations. Base is the published reference from Table 1; +V denotes full VerMem.

| Configuration | ALFWorld | SciWorld | HotpotQA $100J_{\mathrm{judge}}$ |
|---|---|---|---|
| Base | 27.16 | 13.80 | 38.36 |
| +LT | 37.73 | 27.98 | 45.76 |
| +LT/ST | 39.02 | 31.26 | 48.73 |
| +noV | 40.18 | 33.94 | 53.08 |
| +V | 46.30 | 43.39 | 62.75 |

The successive configurations have higher reported scores on all three tasks. The local/composite-global branch comparison in Table 3 instead varies the feedback configuration. Its combined branch scores above either single branch; this observation concerns the combined feedback configuration, including the programmatic terms in the global branch.

# F VERIFIER, CREDIT, REWARD, AND REPRODUCIBILITY SPECIFICATIONS

Command semantics and completed-task utility enter learning through different scoring paths. The local verifier evaluates one realized transition, whereas the global verifier contributes two semantic scores to a composite trajectory reward. Their normalized advantages combine with direct execution costs in the command-token objective.

## F.1 VERIFIER INSTANTIATION AND SCORING PROTOCOL

The local and global verifiers use frozen DeepSeek-V3.2 instances with separate prompts and input scopes. The local verifier receives the task $q$, the bounded policy-visible state $x_t = g(s_t)$, the complete command $u_t$, the realized memory transition, and the source evidence visible at the current decision. It does not receive an unbounded concatenation of $H_t$, nor does it observe later memory decisions, the final answer, or the global-verifier output. The global verifier is invoked after trajectory termination. Its input contains the task, reference answer, annotated supporting facts, final answer, ordered memory commands, their realized transitions, the terminal $M_t$, $C_t$, and $H_t$. Online cost is computed programmatically and is not provided as a verifier-produced score.

The local verifier has an $8{,}192$-token input capacity, separate from the policy-state serialization limit. The global verifier has a $16{,}384$-token input capacity. Its first budget-reduction step removes raw task-solver spans referenced by neither memory commands nor the final answer. These capacities apply to verifier inputs, separately from their output limits.

Table 15 records the separate input capacities and shared output and decoding settings.

Table 15: Configuration of the local and global verifiers. The two verifiers use separate prompts and are disabled during evaluation.

| **Setting** | **Local Verifier** | **Global Verifier** |
|---|---|---|
| Model | Frozen DeepSeek-V3.2 | Frozen DeepSeek-V3.2 |
| Input capacity | 8,192 tokens | 16,384 tokens |
| Maximum output | 128 output tokens per scoring call | 128 output tokens per scoring call |
| Decoding | Temperature 0; top-$p$ 1.0 | Temperature 0; top-$p$ 1.0 |
| Output dimensions | Relevance, grounding, local progress, and information fidelity | Evidence coherence and terminal-memory consistency |
| Score range | Integer 0–4 | Integer 0–4 |
| Format retry | One format retry with the same generation settings | One format retry with the same generation settings |

Each verifier dimension is scored with an integer in $\{0, 1, 2, 3, 4\}$. Score $0$ denotes a contradicted or harmful decision; score $1$ denotes weak support with major errors; score $2$ denotes a partially justified decision with clear omissions; score $3$ denotes a sound decision with minor deficiencies; and score $4$ denotes a fully justified decision. Scores are divided by $4$ before reward computation.

The first scoring call and the single permitted format retry use the same output cap and decoding settings in Table 15. The retry retains the original task, state, command or trajectory, evidence, and rubric; it only appends a format reminder and does not change the scoring dimensions. These settings apply only when the corresponding training branch is enabled. Parsing failure is a scoring-interface error and does not add a command violation cost $c_{t,k}$.

**Scoring roles and evidence provenance.** Verifier inputs follow the role-specific permissions in Table 9. Local scoring uses the current command and its realized transition; global scoring uses the completed trajectory and reference labels. Source evidence retains its document title, sentence position, source identifier, and original text as described in Appendix D.1. Supporting-fact recall is computed programmatically, while the global verifier supplies coherence and terminal-memory consistency. The two semantic outputs enter the composite reward through Equation (5). Parsing an invalid scoring response is distinct from rejecting a memory command: the latter changes execution status and the constraint channel, whereas a scoring-format error does not add $c_{t,k}$. Teacher generation and candidate filtering follow Appendices D.2 and D.3.

### F.2 LOCAL VERIFIER

The local verifier evaluates one executable complete command and its realized state transition. Executable commands include valid null decisions, which use the separate null rubric below. It scores task relevance, evidence grounding, local progress, and information fidelity. The four scores are divided by $4$ and averaged to obtain $r_{t,k}^{\text{local}}$. The trajectory-level local reward in Equation (14) sums the scores produced for executable decisions in $\mathcal{E}_k$ and normalizes them by the full memory-decision horizon $T_k$. It is used only for monitoring. It does not replace the operation-wise normalization in Equation (7) used to compute $A_{t,k}^{\text{local}}$.

Relevance evaluates whether the state presents the memory need addressed by the command. Grounding evaluates its support in visible evidence. Local progress concerns improvement in the task or memory state, while information fidelity concerns preservation of valid facts and source attribution.

Table 16 gives the tool-specific conditions for the highest local score. Lower scores reflect increasing departures from these conditions.

Table 16: Operation-specific rubric used by the local verifier. Structural validity and hard constraints are evaluated separately.

| Decision | Condition for a Fully Justified Transition |
|---|---|
| Add | The content has persistent utility, follows visible source evidence, and is not already represented by an active entry in $M_t$. |
| Update | New evidence revises an existing entry, the target is correct, and valid information not contradicted by the new evidence is preserved. |
| Delete | The target is incorrect, unsupported, duplicated, or obsolete, and soft deletion does not remove information that remains necessary for the task. |
| Retrieve | The returned active LTM entries fill a current information gap in $C_t$ without introducing substantial irrelevant or outdated content. |
| Filter | The removed items are irrelevant or redundant, while the task instruction, current environment state, and indispensable evidence remain in $C_t$. |
| SelectEpisode | The selected fragments belong to the current task history and restore earlier observations, tool outputs, or intermediate conclusions required by the current task state. |
| Summarize | The operation reduces active-context length while preserving critical facts, relations, and source attribution. |
| $\varnothing$ | The current state requires no memory transition. The null command leaves $M_t$ and $C_t$ unchanged. The identity decision is retained in the rollout record and does not create a memory-state modification. |

Commands rejected by the execution checks in Appendix A are not sent to the local verifier. Their $A_{t,k}^{\text{local}}$ is set to zero. They still receive the trajectory-level $A_k^{\text{global}}$, while the corresponding $c_{t,k}$ supplies the direct decision-level penalty.

The following block gives the local verifier prompt. The operation-specific condition is selected from Table 16.

**Local-verifier prompt.**

```
You are the local verifier used during VerMem training.

Evaluate exactly one realized atomic memory transition.
The transition has already passed the structural,
reference, precondition, and budget checks.

Use only the task, the state before the command, the
selected command, the realized state after the command,
and the visible source evidence.

Do not use a later trajectory state, the final answer, or
information that is absent from the supplied state.

Score four dimensions with integers from 0 to 4.
```

```
0: contradicted, harmful, or entirely unjustified
1: weakly supported with major errors
2: partially justified with clear omissions
3: sound with minor deficiencies
4: fully justified for the current state

RELEVANCE
Does the command address the current memory need?

GROUNDING
Is the command supported by visible source evidence?

LOCAL_PROGRESS
Does the realized transition improve the current task or
memory state?

INFORMATION_FIDELITY
Does the transition preserve the valid information required
from its sources?

Output exactly four lines and no additional text:

RELEVANCE: <0-4>
GROUNDING: <0-4>
LOCAL_PROGRESS: <0-4>
INFORMATION_FIDELITY: <0-4>

Task q:
[TASK]

Bounded state before the command x_t:
[STATE_BEFORE]

Complete memory command u_t:
[OPERATION]

State after the command:
[STATE_AFTER]

Visible source evidence:
[VISIBLE_EVIDENCE]

Operation-specific condition:
[OPERATION_CONDITION]
```

### F.3 Composite Global Branch and Global Verifier

The composite global branch follows Equations (4) and (5). It combines task correctness, supporting-fact recall, evidence coherence, terminal-memory consistency, and trajectory efficiency. The global verifier produces only evidence coherence and terminal-memory consistency; the other components are programmatic. The four resulting reward terms in Equation (4) are normalized to $[0, 1]$ and averaged with equal weight.

VerMem-noVerify uses the normalized task-outcome advantage and hard constraints. It does not use the local operation advantage or the composite global trajectory advantage.

For HotpotQA, $r_k^{\text{task}}$ is the mean of exact match and token-level F1 against the reference answer. Both values are normalized to $[0, 1]$. If no final answer is produced, $r_k^{\text{task}}$ is set to zero.

The evidence component uses both annotated supporting facts and global verification. Supporting-fact annotations are not exposed to the memory operation policy. They are used for training-instance construction, programmatic supporting-fact recall, verifier inputs, and evaluation. The supporting-fact recall is averaged with the normalized evidence-coherence score returned by the global verifier

to obtain $r_k^{\text{evid}}$. Evidence coherence measures whether the final answer follows from a complete, task-relevant, and non-contradictory evidence chain.

The global verifier also returns terminal-memory consistency. Its normalized score forms $r_k^{\text{state}}$. This dimension checks whether the terminal LTM, active context, and episodic history remain consistent with the observed evidence, whether critical supporting facts remain accessible, and whether obsolete or conflicting entries have been handled. Because $H_t$ is an ordered provenance record, superseded, soft-deleted, rejected, or failed events may remain in history when their status is explicit.

The normalized online cost $\overline{C}_k^{\text{online}}$ uses online tokens, task steps, and memory-tool calls. The three costs are independently min–max normalized within the $K = 8$ candidate trajectories sampled for the same task and then averaged. Each cost component is set to zero when all candidate trajectories have the same value for that component. Teacher calls, training local/global verifier calls, offline Qwen-Max evaluation, and offline index construction are excluded. Any model call made during online execution contributes to online cost, including a semantic execution check if one is used.

The completion threshold is $\delta = 0.5$, as used in Equation (6). The efficiency component is available only when $r_k^{\text{task}} \geq \delta$. The gate is applied to the mean of EM and token F1. It excludes scores below 0.5 rather than all non-exact answers; EM equal to zero with token F1 equal to one reaches the threshold.

The global-verifier prompt confines semantic scoring to evidence coherence and terminal-memory consistency. Tool syntax, required arguments, hard preconditions, and the programmatic reward components are evaluated separately.

**Global-verifier prompt.**

```
You are the global verifier used during VerMem training.

Evaluate the completed task trajectory as a whole.

Answer correctness, supporting-fact recall, online cost,
tool syntax, and hard preconditions are computed
separately. Do not rescore these items.

Score two dimensions with integers from 0 to 4.

0: contradicted, incoherent, or seriously inconsistent
1: major evidence or terminal-memory failures
2: partially coherent with important omissions
3: coherent with minor deficiencies
4: complete, well supported, and internally consistent

EVIDENCE_COHERENCE
Does the final answer follow from a complete,
task-relevant, and non-contradictory evidence chain?

TERMINAL_MEMORY_CONSISTENCY
Are the terminal long-term memory, active context, and
episodic history consistent with the observed evidence?
Are critical facts preserved and accessible? Are obsolete
or conflicting entries handled appropriately? Historical
superseded, deleted, rejected, or failed events may remain
when their status is explicit.

Output exactly two lines and no additional text:

EVIDENCE_COHERENCE: <0-4>
TERMINAL_MEMORY_CONSISTENCY: <0-4>

Task q:
[TASK]

Reference answer:
```

```
[REFERENCE_ANSWER]

Annotated supporting facts:
[SUPPORTING_FACTS]

Final answer:
[FINAL_ANSWER]

Ordered memory commands and realized transitions:
[MEMORY_TRAJECTORY]

Terminal long-term memory:
[TERMINAL_LTM]

Terminal active context:
[TERMINAL_CONTEXT]

Terminal episodic history:
[TERMINAL_HISTORY]
```

### F.4 Advantage Normalization and Hierarchical Credit

The composite global score is normalized within the $K = 8$ trajectories sampled from the same task. Local scores are normalized by operation type $v_{t,k}$, as defined in Equation (7). The seven atomic tools and the null type $v = \varnothing$ maintain separate local statistics.

When an operation type appears at least eight times in the current update, its current-update mean and standard deviation are used. When fewer than eight instances are available, phase-specific running statistics are used. Before each phase, the running statistics are initialized from $128$ executable development transitions for every decision type enabled in that phase. The statistics are updated with an exponential moving-average decay of $0.99$. They are reinitialized at the start of every phase.

Both local and global normalization use a standard-deviation floor of $0.01$, and the $\epsilon$ in the main paper is set to $10^{-6}$. If every trajectory in a candidate group has the same composite global score, all corresponding global advantages are zero. Local and global advantages are clipped to $[-5, 5]$ before they are combined.

The three coefficients retain the unit values specified with Equation (8). Each rejected command receives $A^{\text{local}}_{t,k} = 0$. Its global advantage remains available because the completed trajectory still determines $A^{\text{global}}_{k}$. The constraint cost is assigned directly to the corresponding memory decision.

Table 17 summarizes the fixed normalization and optimization settings.

### F.5 Auxiliary Constraints

The constraint channel assigns costs to specified execution violations. Table 18 reports the fixed costs and execution behavior. Multiple violations at one memory decision are capped at $c_{t,k} = 1.0$. The valid null decision counts toward the memory-decision horizon used by the auxiliary cost, but it is not a memory-tool call and does not incur a tool-call cost.

Evidence grounding and information fidelity remain graded semantic scores. The execution contract separately disallows unsupported writes, unjustified deletion, and loss of indispensable evidence. Retention in $H_t$ and recovery within the remaining decision and token budgets are different properties; membership in $M_t$ or $H_t$ alone does not decide admissibility. Appendix A.4 describes the execution contracts and corresponding state transitions. Partial omissions in executable commands instead affect the local score.

### F.6 Token-Level GRPO Implementation

For command tokens $y_{t,k,1:L_{t,k}}$, let $h_{t,k,j} = (x_{t,k}, y_{t,k,<j})$. The ratio $\rho_{t,k,j}$ compares the current policy to the policy frozen before the update. The KL term $d^{\text{KL}}_{t,k,j}$ compares it to a reference policy

Table 17: Fixed normalization, credit-assignment, and stabilization settings used by VerMem.

| Setting | Value | Setting | Value |
|---|---|---|---|
| Candidate-group size $K$ | 8 | Minimum current-update count | 8 |
| Running-statistic initialization | 128 per decision | Running-statistic decay | 0.99 |
| Standard-deviation floor | 0.01 | $\epsilon$ | $10^{-6}$ |
| Advantage clipping | $[-5, 5]$ | $\delta$ | 0.5 |
| $\lambda_{\text{local}}$ | 1.0 | $\lambda_{\text{global}}$ | 1.0 |
| $\lambda_{\text{constraint}}$ | 1.0 | $\epsilon_{\text{clip}}$ | 0.2 |
| $\beta_{\text{KL}}$ | 0.1 | Entropy coefficient | 0 |

Table 18: Constraint costs and execution behavior. Costs from multiple violations at one memory decision are capped at 1.0.

| Violation | $\mathbf{c_{t,k}}$ | Execution Behavior |
|---|---|---|
| Invalid tool, missing argument, invalid reference, or failed precondition | 1.0 | Reject the transition, keep $M_t$ and $C_t$ unchanged, record the failed command and reason in $H_t$, and continue the task. |
| Unsupported Add or Update, or an unjustified Delete | 1.0 | Reject the transition, keep $M_t$ and $C_t$ unchanged, and record the failure in $H_t$. |
| Unrecoverable removal of indispensable evidence | 1.0 | Reject the proposed transition and preserve the previous $M_t$ and $C_t$; record the failure in $H_t$. |
| Active-context overflow | 0.5 | Reject the transition, keep $M_t$ and $C_t$ unchanged, and record the failed command and status in $H_t$. |
| Memory-decision or interaction-limit violation | 1.0 | Terminate the trajectory at the configured limit. |
| Reference-answer or future-information leakage | 1.0 | Terminate the trajectory and set its task and evidence components to zero. |

fixed within the phase:

$$\rho_{t,k,j}(\theta) = \frac{\pi_\theta(y_{t,k,j} \mid h_{t,k,j})}{\pi_{\theta_{\text{old}}}(y_{t,k,j} \mid h_{t,k,j})}, \qquad d^{\text{KL}}_{t,k,j} = D_{\text{KL}}(\pi_\theta(\cdot \mid h_{t,k,j}) \,\|\, \pi_{\text{ref}}(\cdot \mid h_{t,k,j}))\,. \tag{18}$$

The old policy supplies the likelihood denominator for the current update. The reference anchors optimization throughout a phase, using $\theta_{\text{SFT}}$, $\theta_A$, and $\theta_B$ for Phases A, B, and C, respectively. Thus, the old policy follows the updates, whereas the reference changes at phase boundaries. Both distributions are evaluated at the same serialized state and generated command prefix.

The policy ratio is evaluated token by token as $\rho_{t,k,j}(\theta)$ in Equation (18). All generated tokens that serialize one complete command $u_{t,k}$ share the same $A^{\text{hier}}_{t,k}$. The token loss of one command is averaged over its command tokens before losses are averaged over $\mathcal{B}_{\text{op}}$. This gives each command equal weight before batch averaging, independent of argument length.

The loss mask includes only the complete command tokens $y_{t,k,1:L_{t,k}}$, including the operation type and structured arguments. The task specification, $M_t$, $C_t$, $H_t$, system instructions, task-solver outputs, environment observations, and padding tokens are excluded. The null action $\varnothing$ remains a trainable memory decision, and it counts toward the memory-decision horizon. It is not counted as a memory-tool call.

The objective uses $\epsilon_{\text{clip}} = 0.2$ and $\beta_{\text{KL}} = 0.1$, as specified in Equation (9). The reinforcement-learning rate is $10^{-6}$, the maximum gradient norm is 1.0, and no additional entropy bonus is used. Local and global advantages are clipped to $[-5, 5]$.

### F.7 Reward Monitoring Protocol

Answer-Only and All-Returns use $m_k^{\text{ans}}$ and $m_k^{\text{all}}$ from Equations (17) and (16). The local monitor aggregate divides by the full memory-decision horizon $T_k$; operation-wise advantage normalization uses its own reference set. Each panel therefore tracks the evolution of one strategy-specific monitor. Cross-strategy task and resource comparisons use the common held-out metrics in Table 4.

Figure 7 shows the mean monitors during HotpotQA training with Qwen2.5-7B-Instruct. The schedule contains $3{,}500$ policy updates and $101$ records. Record $0$ corresponds to the rollout batch before the first update; subsequent records are separated by $35$ updates. Each record first averages the monitor over the current batch of $64$ candidate trajectories, then averages across seeds $42$, $43$, and $44$. The horizontal axes show cumulative policy updates. Phase A, B, and C comprise $1{,}000$, $1{,}000$, and $1{,}500$ updates, respectively.

The lines connect the recorded means without a moving average, temporal smoothing, or spline interpolation. These monitors are separate from the normalized policy advantages, training loss, and checkpoint-selection criteria. Their different reward mappings give the panels different measurement scales despite the shared numerical range $[0, 1]$. Table 4 evaluates answer quality, memory quality, actual online tokens, and tool calls on common scales.

### F.8 Protocol-Level Behavior and Failure Analysis

The scenarios in Tables 19–21 connect memory needs to state transitions and training credit. They are protocol-level illustrations, rather than sampled rollout records.

#### F.8.1 Phase-Specific Memory Decisions

The three training phases expose different memory needs. Phase A presents incremental facts, corrections, duplicate content, and obsolete entries. Visible evidence determines whether an LTM entry should be added, updated, soft-deleted, or retained.

Phase B presents a prepared LTM state, a distractor-rich $C_t$, and same-task history in $H_t$. It requires the policy to distinguish Retrieve from SelectEpisode and Filter from Summarize. Phase C combines both tool groups in one trajectory. Early Add or Update decisions may support a later Retrieve, while Filter, Summarize, or SelectEpisode changes the evidence visible to the task solver.

Table 19 summarizes the decision patterns that define these scenarios. The state effects follow the tool semantics in Appendix A.

#### F.8.2 Historical-Context Recovery

The historical-context recovery scenarios in Phase B make the distinction between $M_t$ and $H_t$ explicit. The initial $C_0$ is distractor-rich, while $H_0$ contains same-task history. Retrieve cannot recover evidence that was never written to $M_t$. SelectEpisode instead identifies the relevant same-task fragment in $H_t$ and restores it to $C_t$, subject to the configured active-context budget.

Filter may then remove distractors, while Summarize may compress the restored evidence before the task solver continues. This sequence recovers earlier task evidence without treating episodic history as another persistent memory store.

During Phase C, historical-context recovery can occur between LTM maintenance and final reasoning. Early Add or Update decisions may preserve persistent knowledge, while a later SelectEpisode restores an observation or tool output that was useful only after the task focus changed. This training scenario requires coordination of $M_t$, $C_t$, and $H_t$.

#### F.8.3 Interpretation of Local and Composite-Global Credit

The two credit branches answer different questions. The local verifier evaluates whether the current executable command $u_t$ is relevant, grounded, useful, and faithful in $x_t$. The composite global branch combines task correctness, supporting-fact recall, efficiency, and the global verifier's evidence-coherence and terminal-memory-consistency scores. Table 20 explains their signs under explicit relative-score conditions.

Table 19: Qualitative decision patterns covered by the three-stage training curriculum. Each row links a memory need to the corresponding atomic operation, realized state effect, and targeted failure mode.

| **Memory Need** | **Decision** | **Realized State Effect** | **Targeted Failure Mode** |
|---|---|---|---|
| New information has persistent value and is absent from active LTM. | Add | Create a source-bearing entry in $M_t$. | Loss of information needed when the known question is solved later. |
| New evidence corrects or refines an active LTM entry. | Update | Create a revised version while preserving the previous version. | Stale or conflicting persistent memory. |
| The active entry is unsupported, obsolete, or duplicated. | Delete | Soft-delete the target while retaining its history. | Repeated retrieval of invalid or redundant content. |
| Required information is present in $M_t$ but absent from $C_t$. | Retrieve | Introduce the relevant active LTM content into $C_t$. | Reasoning without available persistent evidence. |
| The active context contains irrelevant or redundant distractors. | Filter | Remove selected low-value items from $C_t$ while preserving their history in $H_t$. | Context pollution and unnecessary online tokens. |
| Required same-task evidence exists only in earlier $H_t$. | SelectEpisode | Restore selected historical fragments to $C_t$. | Failure of a recent-window or LTM-only recovery strategy. |
| Relevant context is too long for efficient continued reasoning. | Summarize | Replace selected content in $C_t$ with a shorter faithful representation. | Context overflow or repeated processing of the same evidence. |
| The current state has no valid memory need. | $\varnothing$ | Preserve $\widetilde{s}_t = s_t$ and store the decision in the independent rollout record. It counts toward the memory-decision horizon, but not memory-tool calls or TC; its generated tokens contribute to TN. | Unnecessary operations and additional tool cost. |

Table 20: Illustrative credit conditions. Signs depend on scores relative to their normalization references, not on absolute correctness. The final advantage also includes direct constraint cost.

| Relative score condition | Branch signs | Credit interpretation |
|---|---|---|
| Executable command above its operation reference mean; trajectory above its task-group mean. | Local positive; global positive. | Both branches add positive terms; the total still subtracts $c_{t,k}$. |
| Command above its operation mean; trajectory at or below its group mean. | Local positive; global zero or negative. | The command retains a positive local term. The combined sign depends on all terms. |
| Command at or below its operation mean; trajectory above its group mean. | Local zero or negative; global positive. | Local feedback can offset shared positive global credit. |
| Rejected command. | Local zero; global trajectory-dependent. | Preserve $M_t, C_t$, retain global advantage, and subtract the direct cost. No fixed total sign follows. |

The second and third rows show how local credit can differ from the credit shared by a trajectory. Commands above their operation-reference mean retain a positive local term even in a below-average trajectory. Conversely, commands below that mean can offset positive trajectory credit. Operation-wise normalization compares each command with its own tool type's score distribution.

### F.8.4 FAILURE MODES

Table 21 illustrates responses under the tool contracts and constraint channel. Semantic deficiencies in executable commands affect $r_{t,k}^{\text{local}}$; a command rejected by the execution predicate instead receives direct cost $c_{t,k}$.

Table 21: Representative failure modes and their treatment by local semantic verification, composite-global credit, and hard constraints.

| **Failure** | **Risk or Observable Consequence** | **Training Response** | **Execution Behavior** |
|---|---|---|---|
| Duplicate Add | If committed, equivalent information would be stored more than once and could later introduce redundant context. | The rejected command has $A_{t,k}^{\text{local}} = 0$, retains $A_k^{\text{global}}$, and receives $c_{t,k} = 1.0$. | Reject the transition, keep $M_t$ and $C_t$ unchanged, and record the failure in $H_t$. |
| Unsupported Update | If committed, the active entry would be revised without visible evidence and could conflict with persistent memory. | The rejected command has $A_{t,k}^{\text{local}} = 0$, retains $A_k^{\text{global}}$, and receives $c_{t,k} = 1.0$. | Reject the transition, preserve the previous version, and record the failure in $H_t$. |
| Unjustified Delete | If committed, useful persistent evidence would become unavailable to ordinary retrieval. | The rejected command has $A_{t,k}^{\text{local}} = 0$, retains $A_k^{\text{global}}$, and receives $c_{t,k} = 1.0$. | Reject the transition, preserve the active entry, and record the failure in $H_t$. |
| Irrelevant Retrieve | Outdated or task-irrelevant LTM content expands $C_t$ and raises online cost. | Low relevance and local progress; the global efficiency term may also reduce the composite global score. No hard cost is applied when the command remains executable. | Commit only when the operation remains valid and within budget. |
| Over-aggressive Filter | If protected evidence were removed, the resulting $C_t$ could leave a later reasoning gap. | The hard-rejected command has $A_{t,k}^{\text{local}} = 0$, retains $A_k^{\text{global}}$, and receives $c_{t,k} = 1.0$. | If the execution predicate rejects the removal, preserve $C_t$ and record the failure in $H_t$. |
| Lossy Summarize | If committed, the shorter context would alter a supported relation, source, or condition needed by the task. | The hard-rejected command has $A_{t,k}^{\text{local}} = 0$, retains $A_k^{\text{global}}$, and receives the configured hard cost. | Reject the transition, preserve $C_t$, and record the failure in $H_t$ when the summary changes the supported meaning. |
| Misleading SelectEpisode | Recent or semantically similar distractors return to $C_t$ instead of the relevant earlier fragment. | Low relevance and local progress; the completed evidence chain may also lower the composite global score. No hard cost is applied when the selected fragments are valid same-task references. | Commit only same-task fragments that fit the active-context budget. |
| Unnecessary operation | The policy adds tool calls and context changes when $\varnothing$ would preserve a valid state. | Low local progress and potentially lower composite trajectory efficiency; no hard cost is applied when the command is executable. | The transition may remain executable; its normalized credit depends on the applicable score statistics. |

Irrelevant Retrieve commands can remain executable and receive a low semantic score, as can minor executable omissions in Filter or Summarize. Rejected commands receive no local semantic evaluation, retain trajectory-level global credit, and incur their direct constraint cost. The resulting total advantage still depends on all applicable terms.