When Does Memory Help Multi-Trajectory Inference for Tool-Use LLM Agents?

Xinzhe Li
RMIT University
xinzhe.li@rmit.edu.au

Yaguang Tao
RMIT University
yaguang.tao@rmit.edu.au

Abstract

Multi-trajectory inference for tool-use LLM agents — generating multiple reasoning attempts and selecting among them — benefits from transferring knowledge across attempts so that later ones avoid the pitfalls of earlier ones. Existing cross-trajectory memory methods (trajectory-level reflection, atomic fact extraction, raw observation injection) are each evaluated under a single inference strategy on a single task, making it unclear whether reported gains reflect properties of the memory abstraction or of the inference method. We propose a unified framework that decomposes memory along two axes — the *scope* of transfer (within an expansion vs. across trajectories) and the *abstraction* of the transferred content — and evaluate four methods under three inference strategies (best-of-N, beam search, MCTS) on four tool-use benchmarks spanning SQL, knowledge-graph, and CLI environments, in a verifier-free setting that matches the deployment regime of practical agents. The experiment matrix identifies the inference method as a confound: the same memory method produces statistically distinct results under different inference strategies on the same examples. Reflection reaches significance only under MCTS (not under best-of-N); within-expansion injection (conditioning each candidate on prior siblings' outcomes) helps only diversity-starved beam search; and atomic fact extraction is accuracy-neutral but shortens trajectories by 19–26% on tasks with reusable environmental structure.

1 Introduction

Multi-trajectory inference — generating many candidate reasoning paths and selecting among them (Yao et al., 2023; Hao et al., 2023; Gui et al., 2024) — consistently improves accuracy over single-trajectory agents on tool-use benchmarks. One way to further improve multi-trajectory inference is to explicitly *transfer information* across attempts (e.g., avoiding repeated mistakes, reusing discovered environmental knowledge, or diversifying exploration), so that later attempts are informed by earlier ones rather than sampling independently.

Memory augmentation provides this information channel. Existing methods, however, span a heterogeneous design space: Reflexion-style trajectory-level summaries of failed attempts (Shinn et al., 2023), LATS-style reflections injected during tree search (Zhou et al., 2024), and Re-TRAC's structured state combining evidence, plans, and failed-attempt records (Zhu et al., 2026). These methods vary in the *abstraction* at which transferred content is represented (strategy-level reflection vs. structured facts), but all operate at the same *scope*: transferring knowledge across complete trajectories. We identify scope as a second axis — within-expansion transfer (conditioning each sibling candidate on prior siblings' outcomes) — and show that the combination of scope and abstraction interacts nontrivially with the inference strategy (best-of-N, beam search, MCTS) and with task properties (serializable vs. non-serializable environment, presence or absence of an inline verifier).

We focus on *tool-use* agents: multi-step interactive reasoning where the agent issues structured calls (e.g., SQL queries, shell commands) to an external environment. This setting supports both memory abstractions (observational feedback for fact extraction; error signals for reflection) and exposes a structural constraint: when the environment is non-serializable (Zainullina et al., 2025) (state cannot be forked), beam search and MCTS are infeasible, leaving memory-augmented best-of-N as the only viable multi-trajectory strategy.

We propose a unified framework that decomposes memory along the scope$\times$abstraction axes, derives four memory methods as concrete instantiations, and evaluates them in an experiment matrix (4 memory methods $\times$ 3 inference strategies $\times$ 4 benchmarks spanning 3 tool-use environ-

ments, minus structurally inadmissible combinations). Our setting is *verifier-free*: the task verifier is offline, applied at evaluation time only, matching the deployment regime of practical tool-use agents. This distinguishes our analysis from prior memory-augmented multi-trajectory work that relies on inline binary verifiers (exact match, unit-test pass) (Shinn et al., 2023; Zhou et al., 2024).

Our contributions:

- A unified scope×abstraction framework for memory in tool-use tree search, from which existing methods and a new one (Raw Sibling) are derived as instantiations (§3).
- A systematic empirical study under a verifier-free, deployment-faithful setup, covering memory × inference × task cells (§4).
- Three findings on when memory helps: (F1) memory's accuracy effect is search-method-dependent — Reflection reaches significance only under MCTS, while cross-sibling injection helps only diversity-starved beam search; (F2) under MCTS on the harder benchmark (KGQA), Reflection and Raw Sibling produce statistically indistinguishable accuracy despite operating on different memory abstractions; (F3) fact extraction is accuracy-neutral but improves *efficiency* on tasks with reusable environmental structure (§5).

2 Background and Related Work

LLM tree search and multi-trajectory inference. Recent work applies search algorithms to LLM inference, generating multiple reasoning trajectories and selecting the best. Tree-of-Thoughts (Yao et al., 2023) uses BFS/DFS with LLM-generated evaluations. RAP (Hao et al., 2023) formulates reasoning as MCTS with an LLM world model. ReST-MCTS* (Zhang et al., 2024) combines MCTS with process reward models for step-level guidance. Best-of-N sampling (Gui et al., 2024) generates best-of-N and selects via a verifier or reward model. Zainullina et al. (2025) formalize non-serializable environments where state cannot be forked, showing that MCTS is infeasible in such settings and proposing trajectory selection as an alternative.

Reflection and verbal memory in LLM agents. Reflexion (Shinn et al., 2023) stores verbal reflections from failed trials in episodic memory to improve subsequent attempts. LATS (Zhou et al., 2024) integrates reflection into MCTS, generating self-reflections after failed rollouts. Re-TRAC (Zhu et al., 2026) compresses each trajectory into a structured state (evidence, uncertainties, plans) and conditions subsequent attempts on it — a unified representation combining our reflection and fact extraction abstractions. RoT (Zhou et al., 2022) extracts guidelines from prior search trees to reduce repeated mistakes in future searches. ExpeL (Zhao et al., 2024) accumulates cross-task insights from success and failure trajectories. R-MCTS (Yu et al., 2025) adds contrastive reflection to MCTS for web navigation agents. MC-DML (Shi et al., 2025) uses MCTS with a dynamic memory library that stores reflections as PUCT priors for text game agents.

Atomic fact extraction. Fact-based memory has been explored in conversational and planning settings. mem0 (Chhikara et al., 2025) extracts and consolidates atomic facts from multi-session dialogues using vector-indexed storage and similarity-based retrieval. Holt et al. (2025) apply atomic fact augmentation to lookahead search in environment-grounded planning (ALFWorld, TextFrozenLake), extracting task-critical facts from interaction trajectories. Our LiTS-Fact adapts the mem0 extraction pipeline to the multi-attempt tool-use search setting and provides the first controlled comparison against trajectory-level reflection.

These works each propose a specific memory mechanism; none systematically compares different memory abstractions (raw observations vs. reflections vs. extracted facts) within the same experimental setup.

Multi-agent memory sharing. Cross-agent memory collaboration has been explored via shared memory pools (Gao and Zhang, 2024), contrastive trajectory distillation across heterogeneous models (Chang et al., 2026), swarm-inspired role specialization (Li et al., 2025), and off-trajectory reasoning where injected partial traces can act as distraction rather than guidance (Li and Goyal, 2026). These approaches use model ensembles; our work studies the same exploration-exploitation tradeoff within a single agent across multiple attempts.

Connection to reinforcement learning. Our framework can be viewed as an inference-time analogue of experience replay (Lin, 1992): processed experience (reflections, extracted facts) is injected into the policy prompt rather than used for gradi-

ent updates. We discuss connections to in-context learning and hindsight experience replay in Appendix A.

3 A Unified Memory Framework

We formalize context augmentation for LLM-based multi-trajectory reasoning. Let $\pi_\theta(a \mid s)$ denote the base policy (an LLM generating candidate actions given state s). A *context augmentor* modifies the policy by conditioning on additional context:

$$\pi_\theta(a \mid s, \mathcal{C}), \quad \text{where} \quad \mathcal{C} = \bigcup_{k=1}^{K} f_k(\mathcal{H}_k). \tag{1}$$

Here f_k is the k-th augmentor's *analyze* function, and $\mathcal{H}_k$ is the history accessible to augmentor k. In practice, $\mathcal{C}$ is concatenated into the LLM prompt alongside s.

Our framework decomposes memory along two orthogonal axes: *scope* (§3.1) — what history is accessible to the augmentor, and *abstraction* (§3.2) — how that history is transformed before injection into the prompt. Concrete memory methods (§3.3) are instantiations of specific (scope, abstraction) pairs, and multiple methods compose naturally via Eq. (1).

3.1 Memory Scope

The memory scope defines $\mathcal{H}_k$ — what history the augmentor can access. Let $\tau_i = (s_0, a_0, s_1, a_1, \ldots, s_T)$ denote the i-th trajectory within a single search, and $\mathcal{T}_n$ the set of all trajectories from the n-th task instance.

Cross-sibling (within expansion). During a single expansion of N candidates at node s, the augmentor provides each candidate with the actions and observations of previously sampled siblings:

$$\mathcal{H}^{(i)}_{\text{sib}} = \{(a_j, o_j)\}_{j<i}, \quad o_j = T(s, a_j), \tag{2}$$

where T is the transition function. This requires *interleaved expansion*: candidates are sampled sequentially, each conditioned on the completed steps of prior siblings:

$$a_i \sim \pi_\theta\big(a \mid s,\ \mathcal{C},\ \{(a_j, o_j)\}_{j<i}\big). \tag{3}$$

Cross-sibling scope is only applicable to inference methods that expand multiple candidates at the same node (beam search, MCTS), not to best-of-N. Appendix B contrasts this interleaved expansion with the standard batch expansion used by methods without cross-sibling scope.

Cross-trajectory (within search). The augmentor accesses trajectories from previous iterations of the same search:

$$\mathcal{H}_{\text{traj}} = \bigcup_{j<i} \tau_j. \tag{4}$$

This includes both step-level fact sharing (retrieving environmental discoveries from prior trajectories) and trajectory-level profiling (analyzing completed trajectories and injecting structured summaries). Cross-trajectory scope applies to all three inference methods: in best-of-N, each attempt is a "trajectory" whose knowledge transfers to subsequent attempts.

A third scope, *cross-task* (across task instances), is the native operating regime of cross-task memory systems such as ExpeL (Zhao et al., 2024) and mem0 (Chhikara et al., 2025). We focus this paper on cross-trajectory memory within a single task instance and leave a systematic study of cross-task memory to future work.

3.2 Abstraction Level

The abstraction level determines how history $\mathcal{H}_k$ is transformed into context $\mathcal{C}_k = f_k(\mathcal{H}_k)$ before prompt injection. We identify three levels, forming a spectrum from zero processing cost to maximum compression:

Raw (no abstraction). The history is injected verbatim — complete action–observation pairs without any LLM summarization. Cost: zero additional LLM calls. Limitation: consumes context window proportional to history size; only practical for small histories (e.g., a few sibling steps).

Reflection (trajectory-level summary). An LLM generates a natural-language summary of an entire trajectory, capturing strategy-level lessons (e.g., "the C compilation approach failed due to missing libraries; try a Python-based solution instead"). Cost: one LLM call per trajectory.

Atomic fact extraction (step-level or batch). An LLM extracts discrete factual statements from observations (e.g., "config file located at `/etc/app/config.yaml`", "`apt-get install gcc` failed: package not found"). Facts are deduplicated against existing entries via embedding-based similarity at write time; at inference, all facts collected from prior trajectories are injected into the policy prompt (we discuss alternative retrieval policies in §6). Cost: one LLM call per step (incre-

Table 1: Memory methods as (scope, abstraction) instantiations. Methods marked (ours) are novel; Reflection corresponds to Reflexion (Shinn et al., 2023) / LATS (Zhou et al., 2024).

Method	Scope	Abstraction
No Memory (baseline)	—	—
Raw Sibling (ours)	within-expansion	raw
Reflection	cross-trajectory	reflection
LiTS-Fact	cross-trajectory	fact extraction

mental) or per trajectory (batch), plus embedding computation for deduplication.

Reflection vs. fact extraction. In best-of-N with full-trajectory processing, reflection and fact extraction share the same pipeline structure (one LLM call on the complete trajectory, results injected into subsequent attempts) but differ in *what* the LLM is asked to produce: reflection yields strategy-level advice, while fact extraction yields observation-level atomic statements. This makes them a controlled comparison of abstraction granularity under identical pipeline conditions.

3.3 Deriving Concrete Methods

Each concrete memory method is an instantiation of a (scope, abstraction) pair. Table 1 shows how the four methods evaluated in this paper are derived from the framework (Appendix C discusses the design rationale for each).

Cross-sibling and cross-trajectory scopes are orthogonal: Raw Sibling provides immediate diversity within a single expansion, while LiTS-Fact provides accumulated knowledge across iterations. Their composition (Eq. (5)) tests whether the two information sources are complementary:

$$a_i \sim \pi_\theta\big(a \mid s,\ \underbrace{\mathcal{C}_{\text{fact}}}_{\text{cross-traj}},\ \underbrace{\{(a_j, o_j)\}_{j<i}}_{\text{cross-sib}}\big). \quad (5)$$

Evaluated cells. The full 2×3 matrix of (scope, abstraction) pairs contains six cells; three are degenerate or undefined (Appendix D). The four methods in Table 1 are the well-defined, non-degenerate instantiations. Not every method applies to every inference method: within-expansion scope requires multi-candidate expansion (beam search, MCTS), and cross-trajectory scope requires multiple iterations. These constraints determine which cells of the experiment matrix (Table 2) are admissible; §3.4 gives the full admissibility rules.

3.4 Task Types

Our study targets tool-use agents (§1): multi-step interactive reasoning where the agent repeatedly issues structured calls (SQL queries, SPARQL queries, shell commands) to an external environment. This includes both single-tool multi-step settings (an SQL agent issuing repeated queries to one database) and multi-tool settings (a CLI agent invoking different shell commands).

Serializable vs. non-serializable environments. A serializable environment allows saving and restoring state (e.g., a read-only database). Tree search and beam search require serializability because they explore multiple branches from the same state. A non-serializable environment (e.g., a Docker container whose filesystem is mutated by each command) cannot be forked, restricting the agent to sequential trajectories. Memory-augmented best-of-N is therefore the only multi-trajectory strategy available in non-serializable settings.

Admissibility of method-search combinations. The unified scope $\times$ abstraction $\times$ inference-method framework determines which combinations are well-defined. *Cross-sibling scope* requires multi-candidate expansion (Beam, MCTS) and is undefined for best-of-N; *cross-trajectory abstractions* (Reflection, Fact memory) require multiple iterations and on 1-iteration Beam Search collapse to their per-trajectory variants in the best-of-N row; *within-trajectory injection* (per-step Fact) generalizes to multi-iteration settings without modification. These constraints carve the experimental matrix (Table 2) into the cells we report and explain the empty cells as inadmissible rather than untested.

4 Experimental Setup

Inference methods. We consider three inference methods that differ in search structure. **Best-of-N:** N trajectories are generated sequentially; memory transforms i.i.d. sampling into sequential attempts conditioned on prior knowledge. This is the only viable multi-trajectory strategy for non-serializable environments (Zainullina et al., 2025) where state cannot be forked (e.g., Terminal-Bench). **Beam search** (Yao et al., 2023)**:** B candidate continuations are scored per step and the top-B retained; memory operates at per-step granularity across beams. **MCTS** (Hao et al., 2023; Zhou et al., 2024)**:** UCT selection, expansion, simulation, and

backpropagation; memory operates at both per-step (expansion) and per-trajectory (backpropagation) granularity. We follow Reflexion's policy-only injection convention across all Reflection cells (Appendix F.5); the LATS variant of injecting reflections into the reward prompt is available under MCTS but is not used in our matrix to keep the per-method comparison controlled. Backpropagation configuration and full search settings are documented in Appendix E and F.

Memory methods. We evaluate the four methods in Table 1: No Memory, Raw Sibling, Reflection, and LiTS-Fact; their scope $\times$ abstraction construction is given in §3.3 and per-method details (triggers, applicability) in Appendix G. LiTS-Fact adapts the atomic-fact extraction and deduplication mechanism of mem0 (Chhikara et al., 2025) (originally designed for cross-session dialogue) to the cross-trajectory scope of multi-attempt search, where the goal is caching reusable environmental knowledge so that subsequent attempts skip redundant discovery steps.

Benchmarks. All benchmarks are tool-use tasks with structured tool calls. **WikiSQL / WikiTQ** (SQL Agent) (Liu et al., 2024): the agent queries a relational database; serializable (read-only state). **KGQA** (KG Agent) (Liu et al., 2024): the agent traverses Freebase via SPARQL for multi-hop questions; serializable. **Terminal-Bench 2.0** (CLI Agent) (Merrill et al., 2026): the agent executes shell commands to complete software engineering tasks; *non-serializable* (state is mutated and cannot be forked), making beam search and MCTS infeasible.

Implementation. All experiments use LiTS (Li, 2026), a modular tree-search framework whose policy, transition, and reward-model components are shared across all configurations. Memory is injected via a single abstraction that hooks into the search loop without modifying other components (formal pipeline in Appendix H); Raw Sibling additionally requires interleaved expansion ordering (Eq. 3).

Models and reporting. We use Claude Sonnet 4.6 as the policy on KGQA (entity resolution and relation selection require strong language ability) and Claude Haiku 3.5 on WikiSQL, WikiTQ, and Terminal-Bench (where Sonnet's greedy baseline is near-ceiling, e.g., 76.5% on WikiSQL, leaving no headroom for memory experiments). The per-step reward model and all augmentor LLMs are Sonnet 4.6 across every configuration, so memory effects within each row of Table 2 are isolated from supervisor-side variation. KGQA cells are reported on a common 69-example subset (60 ReAct-error + 9 stratified-correct, seed 42; Appendix I); all other benchmarks are reported on their full evaluation sets (WikiSQL: 51, WikiTQ: 49, Terminal-Bench: 89).

5 Results and Analysis

We organise the results around three findings that emerge from the experiment matrix in Table 2, supported by McNemar's exact paired-binary test on each method-pair comparison (Appendix J lists raw p-values).[1] **(F1)** Memory's accuracy effect is search-method-dependent: neither Reflection nor Raw Sibling helps on every cell, and the cells where each helps follow a coherent pattern (§5.1). **(F2)** On the harder benchmark (KGQA) under MCTS specifically, the two otherwise-different memory abstractions become indistinguishable in both their effect size and the specific examples they rescue; this convergence does not hold on the SQL benchmarks, where Reflection substantially outperforms Raw Sibling under MCTS, so we report it as a single-cell finding rather than a generalisation (§5.2). **(F3)** Fact extraction is accuracy-neutral but shortens trajectories by 19–26% on tasks with reusable structure; this is the only main result for which we observe a consistent *efficiency* effect rather than an accuracy effect (§5.3).

5.1 F1: Memory's accuracy effect is search-method-dependent

The four method-pair comparisons that reach $p < 0.05$ under McNemar's exact test (Table 2) cluster in two patterns rather than spreading uniformly across cells:

Pattern 1 — Reflection improves accuracy directionally under both inference methods, but reaches significance only under MCTS. Across all seven Reflection-vs-No-Memory cells where the comparison is defined (Table 2), Reflection's effect is directionally positive. However, only two cells reach McNemar significance, and both lie in the MCTS column: MCTS+Reflection on Wik-

[1] Throughout, we report McNemar's discordant pair counts as (b rescues / c regressions): a rescue is a baseline-incorrect example that becomes correct under the treatment, a regression is the reverse.

Table 2: Experiment matrix: memory method × inference method × task type. **Bold**: best accuracy within each inference-method group. Significance markers: ** $p<0.01$, * $p<0.05$ (McNemar's exact test vs. the No-Memory baseline in the same row group); raw p-values in Appendix J. Terminal-Bench is non-serializable (—). Raw Sibling requires multi-candidate expansion (Beam/MCTS). [†]Not evaluated: LiTS-Fact is accuracy-neutral across all other cells (F3); the additional KGQA MCTS run would confirm the same null at high cost.

Inference Method	**Memory Method**	**SQL Agent** WikiSQL	WikiTQ	**KG Agent** KGQA	**CLI Agent** Terminal-Bench
Single Trajectory (ReAct, T=0)		39.2	28.6	13.0	7.9
Best-of-N (pass@5, T=0.9)	No Memory	49.0	40.8	31.9	23.6
	Reflection	**58.8**	**49.0**	**33.3**	**25.8**
	LiTS-Fact	47.1	34.7	31.9	**25.8**
Beam Search	No Memory	27.5	26.5	7.2	—
	Raw Sibling (ours)	**39.2**	30.6	**27.5****	—
MCTS	No Memory	49.0	38.8	23.2	—
	Raw Sibling (ours)	47.1	36.7	**34.8***	—
	Reflection	**60.8***	**44.9**	**34.8***	—
	LiTS-Fact	47.1	38.8	[†]	—

iSQL (6 rescues / 0 regressions, p=0.031*) and on KGQA (10 rescues / 2 regressions, p=0.039*). The corresponding pass@5 cells on the same examples do not reach significance: WikiSQL pass@5+Reflection has 5 rescues and 2 regressions (p=0.453), and KGQA pass@5+Reflection has 5 rescues and 4 regressions (p=1.0). WikiTQ is consistent in direction in both columns but reaches significance in neither (pass@5: 4 rescues / 1 regression, p=0.375; MCTS: 5 rescues / 2 regressions, p=0.453); at b+c values of 5 and 7 the WikiTQ cells are plausibly underpowered, so we read them as neutral rather than confirming or rejecting the asymmetry. On Terminal-Bench (Best-of-N only) Reflection adds 8 rescues and 6 regressions (p=0.791); without an MCTS run the within-benchmark contrast does not exist. No cell exhibits the reverse pattern (a significant pass@N+Reflection gain alongside a null or negative MCTS+Reflection effect), so no cell rejects Pattern 1.

Mechanism: the per-step reward model filters bad reflections. The most plausible explanation is that MCTS's per-step reward model (PRM) evaluates each reflection-induced expansion and prunes paths the reflection incorrectly recommends before they reach a terminal, whereas under pass@N the same bad reflection runs to completion and shows up as a regression. This filtering account predicts that, within a benchmark, MCTS+Reflection should produce fewer regressions than pass@5+Reflection. The prediction holds on WikiSQL (0 vs. 2 regressions) and KGQA (2 vs. 4) and is the proximate cause of MCTS+Reflection reaching significance on these two cells; it does not hold on WikiTQ (2 vs. 1), which we read as evidence that the PRM's filtering is not uniformly effective across tasks rather than evidence that filtering does not occur. We further trace the per-iteration progression on the two WikiSQL examples that are rescued by MCTS+Reflection but not by Indep+Reflection (Appendix K). Both show the same pattern: iteration 0 (when no reflection content has yet been generated) fails on the same path that MCTS+No Memory fails on, and iteration 1 onwards, after reflection content becomes available, the PRM selects a sibling expansion the reflection guides the policy towards. On KGQA the per-iteration trace is less informative because the PRM evaluates each Freebase relation choice in isolation, making iteration 0 q-values vary substantially across runs; we discuss this case in Appendix K.3.

Pattern 2 — Raw Sibling helps where the policy is diversity-starved. The strongest effect in our study is KGQA Beam+Raw Sibling: 27.5% vs. 7.2% for Beam without memory (Δ=+20.3pp, 17 rescues / 3 regressions, p=0.003**). At n_{iters}=1, Beam exposes a policy diversity collapse (Appendix L): at T=0.7 sampling, sibling candidates frequently converge to the same action or apology, leaving the reward model no diverse candidate to rank. Raw Sibling's interleaved expansion (Eq. 3) biases each next sibling away from prior siblings' actions, which on KGQA's wide action space unlocks 17 rescues. The same mecha-

nism produces a +11.7pp gain on WikiSQL Beam (8 rescues / 2 regressions, p=0.109) and +4.1pp on WikiTQ Beam (3 / 1, p=0.625); under MCTS the gain disappears or reverses (−1.9pp WikiSQL, −2.0pp WikiTQ; both NS), because UCT-driven re-expansion already supplies sufficient action variation. Cross-sibling memory is therefore a remedy for low-diversity configurations rather than a universal addition.

5.2 F2: Memory abstractions converge on KGQA MCTS

Pattern 1 from §5.1 extends to a comparison the paper has not made before: under MCTS on KGQA, Reflection and Raw Sibling produce *the same accuracy and almost the same example set*, despite operating on different memory scopes (cross-trajectory vs. cross-sibling) with different abstraction levels (per-trajectory diagnosis vs. raw action–observation injection). Both methods reach 34.8% (24/69) on the 69-example subset, both rescue exactly 10 ReAct-error examples, and both regress on exactly 2 examples; 9 of the 10 rescued examples are shared; each method uniquely rescues exactly one example the other does not. A direct paired test between the two methods has discordant counts b=c=1, p=1.0 — the two are statistically indistinguishable in their accuracy distribution at N=69.

The convergence does not hold on the SQL benchmarks. Under MCTS on WikiSQL, Reflection reaches 60.8% while Raw Sibling reaches 47.1% (Δ=13.7pp); on WikiTQ the gap is 8.2pp. F2 is therefore a single-cell finding. A plausible explanation is that on KGQA's harder subset, any non-i.i.d. context suffices for recovery, so the abstraction choice stops mattering; on SQL where the policy is closer to solving unaided, Reflection's targeted diagnosis makes a difference.

Why the convergence is interesting even as a single-cell finding. Prior work evaluates a single abstraction per study; any paper that ran only one method on KGQA-MCTS would have reported +11.6pp and attributed the gain to its chosen abstraction. The convergence shows that single-abstraction evaluations may attribute accuracy gains to the wrong cause.

5.3 F3: Fact extraction trades accuracy for efficiency

Across all six evaluated cells, LiTS-Fact's accuracy effect is indistinguishable from zero under McNemar's exact test (all $p \geq 0.250$; see Appendix J for per-cell counts). We therefore do *not* make an accuracy claim for LiTS-Fact. A KGQA-specific failure-mode analysis (over-anchoring on prior facts, procedural-instruction injection, and over-generalised negative facts) is reported in Appendix M.

Where Fact memory does have a measurable effect: trajectory length. LiTS-Fact reduces mean trajectory length from 6.1 to 4.9 steps on WikiSQL (−20%) and from 5.8 to 4.7 on WikiTQ (−19%) by allowing later attempts to skip the `sql_db_list_tables` discovery call: the skip rate in attempts 1–4 rises from 4–5% (No Memory) to 70–77% (LiTS-Fact). On KGQA, mean trajectory length drops from 8.9 to 6.6 steps (−26%) for analogous reasons (cached relation discoveries). Figure 1 and Table 10 (Appendix P) show the per-attempt progression and full efficiency breakdown. On KGQA, mean trajectory length drops from 8.9 to 6.6 steps (−26%) for analogous reasons (cached relation discoveries). On Terminal-Bench, where each task is run in a fresh ad-hoc Docker container with no reusable structure across attempts, fact extraction has no analogous mechanism for shortening trajectories: the extracted facts cannot be reused to skip discovery steps because each task's environment is unique. Across these cells, Fact's effectiveness scales with the *environment regularity* (SQL ≈ KGQA ≫ Terminal-Bench), matching the abstraction's intended target: extracted facts capture what the environment contains (schemas, relations) and become useful only when those facts generalise across attempts.

Reflection and Fact memory are not additive. When both augmentors are run together (Best-of-N row, last sub-cell of WikiSQL/WikiTQ), pass@5 falls to 52.9% / 46.9% — below Reflection alone (58.8% / 49.0%) — and the `list_tables`-skip rate falls to 20% / 23% — below LiTS-Fact alone (77% / 70%). The composition therefore loses on both axes simultaneously. A case study (Appendix N) shows the mechanism: the reflection prompt issues an explicit step-by-step plan that overrides the implicit shortcut Fact memory would offer, so the policy follows the plan and ignores the cached facts.

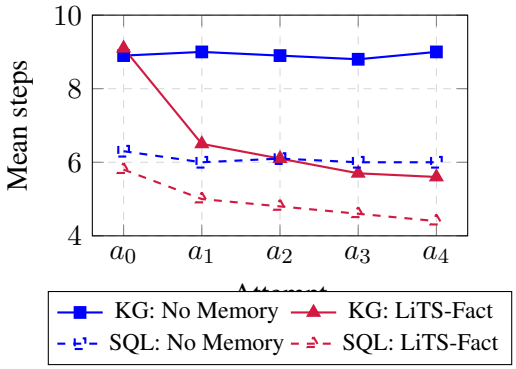

Figure 1: Mean trajectory length by attempt. Baselines (blue) remain flat across attempts — each trajectory starts from scratch. LiTS-Fact (red) decreases as accumulated facts let the agent skip discovery steps. The effect is stronger on KGQA (solid, -3.4 steps by a_4) than SQL (dashed, -1.4 steps), reflecting longer baseline trajectories on KGQA.

6 Discussion

Why Reflection helps only under MCTS: comparison with prior work. LATS (Zhou et al., 2024) and Re-TRAC (Zhu et al., 2026) both evaluate trajectory-level reflection with access to a binary correctness signal (exact match, unit-test pass) during search. Under such signals, it is unclear whether the reported gains come from the reflection content or from the correctness signal preventing bad trajectories from propagating. Our verifier-free setup removes this confound: the only per-step signal available during search is the LLM-based reward model (PRM). The mechanism we propose (§5.1) is that Reflection and PRM act in concert — Reflection provides alternative paths informed by prior failures, and the PRM selects among them — which is why Reflection helps under MCTS (where PRM operates) but not under best-of-N (where no per-step selection exists).

Connection to LLM sensitivity to irrelevant context. The Reflection+Fact composition failure (§5.3) is consistent with reports of LLMs being sensitive to irrelevant context (Shi et al., 2023): once an explicit plan is present, the correct-but-redundant fact memory becomes irrelevant and the policy ignores it, losing on both axes.

7 Conclusion

We proposed a scope $\times$ abstraction taxonomy for cross-trajectory memory in tool-use LLM agents and evaluated four methods under three inference methods on four benchmarks in a controlled matrix. The study identifies the inference method as a confound that must be controlled for in any claim about a memory method's accuracy effect: the same memory mechanism produces statistically distinct results under different inference methods on the same examples. Specifically, reflection reaches significance only under MCTS (not best-of-N), cross-sibling injection helps only diversity-starved Beam Search (not MCTS), and atomic fact extraction is accuracy-neutral but shortens trajectories by 19–26% on tasks with reusable structure. These findings suggest that published claims about cross-trajectory memory effectiveness are not directly comparable across studies, because each evaluates under a different inference method — a confound our controlled matrix makes explicit.

7.1 Reproducibility

We build on the open-source LiTS framework (Li, 2026), extending it with a memory module that implements the four methods in this paper. We will release our fork (memory module + experiment configurations + evaluation scripts) upon acceptance. Per-configuration evaluation results (per-example correctness verdicts) and the analysis scripts used to produce all tables are included in the supplementary material; Appendix Q reports API costs.

Limitations

Our study evaluates a single policy model per benchmark (Haiku 3.5 for SQL tasks, Sonnet 4.6 for KGQA); the diversity dynamics that drive the Raw Sibling and Reflection effects may differ for models with different sampling behaviour at fixed temperature. Sample sizes per cell range from 49 to 89 examples, which is sufficient to detect effects of $\gtrsim$ 6pp under McNemar's exact test but not smaller ones. We evaluate cross-sibling and cross-trajectory memory scopes but not cross-task memory (knowledge transfer across different questions), which may exhibit different dynamics. LiTS-Fact uses non-selective retrieval (all extracted facts are injected); a structural argument for why selective retrieval cannot resolve the diversity–efficiency tradeoff is given in Appendix O.

Ethics Statement

This work involves no human subjects, no personal data, and no deployment of autonomous agents in real-world settings. All experiments use com-

mercial LLM APIs (Anthropic Claude) on publicly available benchmarks. The total API cost is approximately $1,384 (Appendix Q). We do not foresee direct negative societal impacts from the findings; the memory methods studied are general-purpose augmentations to LLM inference and do not introduce new capabilities beyond what the underlying models already possess. We used AI coding assistants (Claude) for code development and paper drafting assistance.

Acknowledgements

This research was supported by compute resources provided by the RMIT Advanced Computing Ecosystem (RACE) through AWS cloud credits.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Yurui Chang, Yiran Wu, Qingyun Wu, and Lu Lin. 2026. Memcollab: Cross-agent memory collaboration via contrastive trajectory distillation. *arXiv preprint arXiv:2603.23234*.

Ziru Chen, Michael White, Ray Mooney, Ali Payani, Yu Su, and Huan Sun. 2024. When is tree search useful for LLM planning? it depends on the discriminator. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13659–13678, Bangkok, Thailand. Association for Computational Linguistics.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.

Hang Gao and Yongfeng Zhang. 2024. Memory sharing for large language model based agents. *arXiv preprint arXiv:2404.09982*.

Lin Gui, Cristina Garbacea, and Victor Veitch. 2024. BoNBon alignment for large language models and the sweetness of best-of-n sampling. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173, Singapore. Association for Computational Linguistics.

Samuel Holt, Max Ruiz Luyten, Thomas Pouplin, and Mihaela van der Schaar. 2025. Improving LLM agent planning with in-context learning via atomic fact augmentation and lookahead search. In *ICML 2025 Workshop on Programmatic Representations for Agent Learning*.

Aochong Oliver Li and Tanya Goyal. 2026. Off-trajectory reasoning: Can LLMs collaborate on reasoning trajectories? In *The Fourteenth International Conference on Learning Representations*.

Ruohao Li, Hongjun Liu, Leyi Zhao, Zisu Li, Jiawei Li, Jiajun Jiang, Linning Xu, Chen Zhao, Mingming Fan, and Chen Liang. 2025. Swarmsys: Decentralized swarm-inspired agents for scalable and adaptive reasoning. *arXiv preprint arXiv:2510.10047*.

Xinzhe Li. 2026. LiTS: A modular framework for LLM tree search. In *ACL 2026 System Demonstration Track*.

Long-Ji Lin. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Mach. Learn.*, 8(3–4):293–321.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2024. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*.

Mike A Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, and 65 others. 2026. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 31210–31227. PMLR.

Zijing Shi, Meng Fang, and Ling Chen. 2025. Monte carlo planning with large language model for text-based games. In *The Thirteenth International Conference on Learning Representations*.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Preprint*, arXiv:2305.10601.

Xiao Yu, Baolin Peng, Vineeth Vajipey, Hao Cheng, Michel Galley, Jianfeng Gao, and Zhou Yu. 2025. ExACT: Teaching AI agents to explore with reflective-MCTS and exploratory learning. In *The Thirteenth International Conference on Learning Representations*.

Karina Zainullina, Alexander Golubev, Maria Trofimova, Sergei Polezhaev, Ibragim Badertdinov, Daria Litvintseva, Simon Karasik, Filipp Fisin, Sergei Skvortsov, Maksim Nekrashevich, Anton Shevtsov, and Boris Yangel. 2025. Guided search strategies in non-serializable environments with applications to software engineering agents. In *Forty-second International Conference on Machine Learning*.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024. ReST-MCTS*: LLM self-training via process reward guided tree search. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Di Zhang, Jianbo Wu, Jingdi Lei, Tong Che, Jiatong Li, Tong Xie, Xiaoshui Huang, Shufei Zhang, Marco Pavone, Yuqiang Li, Wanli Ouyang, and Dongzhan Zhou. 2025. LLaMA-berry: Pairwise optimization for olympiad-level mathematical reasoning via o1-like Monte Carlo tree search. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7315–7337, Albuquerque, New Mexico. Association for Computational Linguistics.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language agent tree search unifies reasoning acting and planning in language models.

Fan Zhou, Haoyu Dong, Qian Liu, Zhoujun Cheng, Shi Han, and Dongmei Zhang. 2022. Reflection of thought: Inversely eliciting numerical reasoning in language models via solving linear systems. *arXiv preprint arXiv:2210.05075*.

Jialiang Zhu, Gongrui Zhang, Xiaolong Ma, Lin Xu, Miaosen Zhang, Ruiqi Yang, Song Wang, Kai Qiu, Zhirong Wu, Qi Dai, and 1 others. 2026. Re-trac: Recursive trajectory compression for deep search agents. *arXiv preprint arXiv:2602.02486*.

A Connection to Reinforcement Learning (Extended)

Our framework is an inference-time analogue of experience replay: instead of storing raw (s, a, r, s') transitions for gradient-based policy updates, we store processed experience (reflections, extracted facts) and inject them into the policy prompt. The "update" is purely in-context — no model parameters change. This places our work in the broader family of in-context learning approaches (Brown et al., 2020) that condition frozen LLMs on task demonstrations or interaction histories, but differs in that we study *what* to store (abstraction level) and *from where* to retrieve (memory scope) rather than how to format demonstrations. Hindsight Experience Replay (Andrychowicz et al., 2017) is a particularly close analogue: it relabels failed trajectories with achieved goals to extract learning signal from failures, paralleling how our reflection augmentor extracts strategy-level lessons from failed attempts.

B Batch vs. Sibling-Aware Expansion

When expanding a node with N candidate actions, the standard approach (*batch expansion*) samples all candidates independently from the same augmented policy:

$$a_1, a_2, \ldots, a_N \overset{\text{i.i.d.}}{\sim} \pi_\theta(a \mid s, \mathcal{C}). \tag{6}$$

This corresponds to the "No Memory" baseline in Table 2 (when $\mathcal{C} = \emptyset$) or to any cross-trajectory memory method without cross-sibling scope (LATS Reflection, LiTS-Fact alone).

Sibling-aware expansion (Eq. (3) in §3.1) replaces i.i.d. sampling with sequential conditioning, creating a dependent chain where each candidate sees the actions and observations of previously sampled siblings. This breaks the i.i.d. assumption but increases action diversity at the cost of sequential latency.

C Method Motivation Details

This appendix expands the brief motivation in §3.3.

Why fact extraction? In tool-use tasks, agents repeatedly discover the same environmental knowledge across trajectories (e.g., querying database schemas, listing available API endpoints). Fact extraction captures this knowledge as reusable atomic statements, potentially saving redundant discovery steps in subsequent trajectories. This is distinct from reflection, which captures strategy-level advice but does not encode concrete environmental state.

Why cross-trajectory fact memory with best-of-N? best-of-N are the only viable multi-trajectory strategy for non-serializable environments (Zainullina et al., 2025), but without memory, each trajectory starts from scratch — repeating the same exploration and making the same mistakes. Cross-trajectory fact memory transforms i.i.d. sampling into sequential refinement: each attempt inherits environmental knowledge (discovered schemas, failed query patterns) from prior attempts, without requiring state serialization.

Why cross-sibling raw context with tree search? In beam search and MCTS, multiple candidates are expanded from the same node. Without cross-sibling context, candidates are sampled i.i.d. from the same prompt, often producing identical actions (action diversity collapse (Li, 2026)). Raw sibling context breaks this degeneracy at zero LLM cost: each candidate sees what its predecessors already tried, naturally diversifying the expansion without requiring explicit diversity penalties or temperature tuning.

D Omitted Cells in the Experiment Matrix

Cells are omitted from Table 2 for two distinct reasons. First, some (scope, abstraction) pairs in the taxonomy are degenerate or undefined, so no concrete memory method exists for them (§D.1). Second, some well-defined memory methods are structurally incompatible with certain inference methods (§D.2).

D.1 Degenerate scope × abstraction cells

The 2×3 taxonomy (scope × abstraction) contains six cells; three are not evaluated:

- *Within-expansion* × *reflection*: reflection requires a complete trajectory to diagnose. Within a single expansion, each sibling has produced only one action–observation pair — insufficient input for a trajectory-level summary.
- *Within-expansion* × *fact extraction*: within a single expansion, each sibling's observation is a single tool return (e.g., one SQL result set, one SPARQL entity list). Extracting "atomic facts" from a single observation yields the observation itself — there is nothing to compress or deduplicate. Fact extraction's value lies in compressing *long* multi-step trajectories (8–15 observations) into a small set of reusable statements; at the single-observation granularity of within-expansion scope, this compression is vacuous and the cell collapses to within-expansion × raw.
- *Cross-trajectory* × *raw*: injecting the full raw history of prior trajectories into the prompt is infeasible for long tool-use trajectories (8–15 steps × ∼500 tokens each × up to 5 prior trajectories exceeds typical context budgets). Reflection and fact extraction exist precisely to compress this history into a manageable representation.

D.2 Inadmissible memory × inference-method cells

Terminal-Bench × Beam Search / MCTS (—). Terminal-Bench uses Docker containers whose state is mutated by each shell command and cannot be forked. Beam search and MCTS require state serialization to explore multiple branches from the same node, making them structurally infeasible in this environment (§3.4).

Beam Search × Cross-Trajectory Memory. Our beam search runs a single round of expansion and selection (no further iteration over the resulting beam). Cross-trajectory memory (reflection, fact extraction) generates context only after a complete trajectory finishes, so any benefit would accrue to a subsequent iteration; with no subsequent iteration, these cells are structurally equivalent to No Memory.

KGQA × Fact + Reflection. On the 69-subset, KGQA Indep+Reflection is essentially flat (31.9% → 33.3%, 5 rescues / 4 regressions, p=1.0; §5.1).

Adding reflection to fact extraction therefore cannot improve over fact extraction alone — it can only add cost or introduce the abstraction-level conflict observed on WikiSQL/WikiTQ (Appendix N). We omit this combination.

Raw Sibling $\times$ best-of-N ($\varnothing$). Raw Sibling requires multi-candidate expansion at the same node. best-of-N generate one trajectory at a time with no shared expansion step, making cross-sibling context inapplicable.

E Backpropagation Configuration

The backpropagation mode in our MCTS rows is determined by two factors: reward source and cross-rollout aggregation.

Reward source. All our benchmarks are tool-use tasks where no objective terminal signal is available (unlike game-playing MCTS where a win/loss outcome is propagated). Each node maintains its own per-step LLM reward aggregated from its position to the leaf.

Cross-rollout aggregation. For non-memory baselines (standard MCTS), rollouts are i.i.d., so we use cumulative backpropagation: each rollout appends a value and Q is the running mean. For memory-augmented configurations, successive iterations are non-independent (later rollouts benefit from accumulated memory context), so we adopt the decay-based update from Zhang et al. (2025): $Q(\text{parent}) \leftarrow (1-\gamma)\,Q(\text{parent}) + \gamma\,Q(\text{child})$, which assigns higher weight to more recent, better-informed trajectories.

F Search Settings: Adapting Reflexion and LATS to Verifier-Free Tool Use

This appendix documents how the **Reflection** cells of Table 2 are configured under our verifier-free deployment-faithful setup. Two prior methods are direct precursors: Reflexion (Shinn et al., 2023) for the best-of-N cell, and LATS (Zhou et al., 2024) for the beam-search and MCTS cells. We share their core mechanism — a per-trajectory LLM reflection that is stored in a persistent buffer and injected into subsequent trajectories — but adapt three settings to a setup without an inline binary task verifier. The same three adaptations apply to both precursors, so we describe them once.

F.1 Setting Choices Compared with Reflexion and LATS

Inheritance. The mechanism in our Reflection cells is unchanged from these precursors: a trajectory-level reflection is generated after a failed trial, stored persistently, and injected into subsequent trials. Reflexion explicitly allows its self-reflection model M_{sr} to be a distinct LLM from the actor M_a (Shinn et al., 2023), so our use of a stronger augmentor (Sonnet) on top of a weaker policy (Haiku) is within the published Reflexion framework, not a deviation.

F.2 No Stop-on-Correct under a Verifier-Free Setup

Both Reflexion and LATS terminate the search the first time a candidate trajectory is judged correct by a binary signal: exact match against gold answers (HotpotQA), unit-test pass (HumanEval), or task-specific heuristics (AlfWorld). Our tool-use benchmarks (SQL, KG, CLI) do not expose such a signal during search: SQL execution-match against the gold answer, SPARQL answer-match, and Terminal-Bench test suites are evaluation-time signals computed by LITS-EVAL, not search-time signals available to the agent. Wiring these signals into the search loop would require the agent to access dataset annotations (gold answers, gold SPARQL outputs, hidden test cases) that are unavailable at deployment, undermining the benchmark's deployment-faithfulness.

The remaining option — stop-on-correct gated by the LLM-based PRM that already guides expansion — is incompatible with the goals of this study for two reasons.

Self-validating signal. If the same PRM both (a) ranks candidate continuations during expansion and (b) decides when the search has succeeded, the search stops the moment the PRM's value estimate clears its own threshold, regardless of whether the answer is actually correct. The benefit of the search — averaging out PRM noise across multiple candidate trajectories — is forfeited precisely when a noisy false-positive estimate triggers early termination.

Mechanism observation. The contribution of this paper is mechanism-level: we study how cross-trajectory memory (reflection, fact extraction) accumulates across multiple completed trajectories within a single search. Stop-on-correct at the first attempt collapses the search into a single-trajectory

Table 3: Settings used for the Reflection cells, compared with Reflexion (best-of-N) and LATS (tree search).

Aspect	Reflexion	LATS	Ours
Inference	ReAct trials	MCTS / beam	both
Trigger signal	EM vs. gold (HotpotQA), heuristics (AlfWorld), unit tests (HumanEval)	EM / unit tests	continuous PRM threshold (r<0.3)
Stop-on-correct	yes (Evaluator passes)	yes (binary verifier)	no
Reflection injection	policy prompt	policy and reward prompts	policy prompt only
Reflection LLM	free choice (M_{sr} separate)	not specified	free choice (Sonnet on Haiku)
Iteration / trial budget	loop until pass or max	search until solved	fixed budget

regime in which neither reflection nor fact extraction has the opportunity to be triggered, retrieved, or composed. The resulting accuracy comparison would not reflect the intended experimental variable.

We therefore disable stop-on-correct and run a fixed iteration / trial budget (§F.4) on every example, deferring verification to LITS-EVAL after the search.

F.3 Continuous PRM Trigger

For the same reason — no inline gold signal — we cannot use Reflexion's binary EM trigger or LATS's binary verifier trigger to gate reflection generation. We trigger reflection on the same PRM score used for search guidance: trajectories with $r < 0.3$ generate reflections; higher-scored trajectories do not. The threshold is set once and held fixed across all benchmarks. This converts the binary trigger into a continuous one, consistent with the rest of our verifier-free setup.

F.4 Fixed Iteration Budget

In place of stop-on-correct we use a fixed budget N_{budget}, i.e., n_{iters}=N_{budget} for MCTS and beam, or N_{budget} pass@N trials for best-of-N. A fixed budget is compute-bounded (per-example cost is predictable and capped) and orthogonal to the PRM's value estimate, avoiding the self-validation issue above. It preserves a multi-trajectory regime in which cross-trajectory memory has the opportunity to act.

The budget value is chosen as a compromise between two pressures. A small budget (e.g., N_{budget}=1) recovers a single-trajectory regime that defeats the purpose of using a multi-trajectory method at all. A large budget (e.g., N_{budget}=30) maximizes mechanism observability but multiplies cost roughly $N_{\text{budget}}\times$, dominated by per-iteration PRM evaluation. We use N_{budget}=5: enough trajectories per example to expose memory dynamics while keeping per-example cost within an order of magnitude of best-of-N pass@5.

F.5 Reflection Injection Site

The original LATS injects reflections from failed prior trajectories into both the policy prompt (steering the next rollout) and the value prompt (informing the reward model's scoring of the current trajectory). Reflexion injects only into the policy prompt. We follow Reflexion's convention — injection into the policy prompt only — across all Reflection cells in our matrix.

The reason is a controlled comparison with LiTS-Fact: under our framework, the two methods share an identical pipeline (per-trajectory trigger, persistent storage, retrieval-and-injection into the policy prompt) and differ only in *what the LLM produces* — strategy-level advice versus observation-level atomic facts. Adding reflection-into-reward to one method but not the other would entangle two variables (abstraction granularity and number of injection sites) and break the controlled study.

A variant injecting reflections into both the policy and the reward prompt (as in the original LATS) is not evaluated in the main matrix. This variant studies a complementary question — whether providing reflection text as additional context to the reward model yields further gains beyond reflection-into-policy alone — which is orthogonal to the abstraction-granularity comparison that motivates the main matrix.

F.6 Why these Differences are not Confounds

The three operational adaptations (continuous PRM trigger, no stop-on-correct, policy-only injection) shift the cell uniformly across all four memory methods evaluated under each inference strategy: No Memory, Raw Sibling (where applicable), Reflection, LiTS-Fact, and Fact + Reflection all use the same trigger, the same budget, and the same injection site. Memory effects reported in Table 2

are measured relative to the No Memory baseline under the same protocol, not relative to Reflexion's or LATS's published numbers, and are not confounded by these protocol changes. For the same reason, our **best-of-N × Reflection** cell can be read directly as the closest correspondence to Reflexion in our matrix, with the three adaptations above as the only differences.

F.7 Hyperparameter Summary

The full set of search hyperparameters used in the Reflection cells of Table 2: n_{actions}=3, n_{iters}=5, exploration constant w_{exp}=1.0, simulate-strategy = max, maximum step depth = 15, rollout depth = 15. Memory-augmented runs use cumulative backpropagation; the decay variant (§E) is reserved for runs where prior iterations' value estimates are explicitly de-weighted in favor of memory-informed later iterations.

F.8 Asymmetric Policy / Supervision Models

Throughout the WikiSQL and WikiTQ MCTS rows of Table 2 we use Haiku 3.5 as the policy LLM and Sonnet 4.6 for all "supervision-side" roles: the reward model, the reflection generator, and the fact extractor. On KGQA, Sonnet 4.6 serves as both policy and supervisor (entity resolution and relation selection require stronger language ability; see §4). The asymmetry — weaker model for action generation, stronger model for evaluation and memory distillation — is deliberate and consistent with Reflexion's modular framework, in which the actor M_a and the self-reflection model M_{sr} are explicitly allowed to differ (Shinn et al., 2023).

Why this asymmetry does not confound the memory deltas. A stronger supervisor evaluating a weaker policy raises the absolute accuracy level reported in the MCTS row, since Sonnet's reward signal is more permissive than Haiku's would be. Memory effects in Table 2 are reported as deltas from a No Memory baseline that uses the *same* Sonnet supervisor across the row (§F.8 below). The level inflation is identical across the No Memory and memory-augmented cells, so it cancels in any pairwise comparison; what remains is the contribution of the memory mechanism itself.

G Memory Methods: Per-Method Details

We give the per-method specification used in our experiments.

No Memory (baseline). Standard inference without any cross-sibling or cross-trajectory information sharing. In beam search and MCTS, candidates at the same node are sampled i.i.d. from the same prompt (batch expansion, Eq. (1) with $\mathcal{C} = \emptyset$).

Raw Sibling (ours). Cross-sibling scope with raw abstraction. Interleaved expansion (Eq. (3)): each candidate sees the complete action–observation pairs of previously sampled siblings, injected as a diversity prompt. No LLM summarization; zero additional LLM calls. Only applicable to beam search and MCTS.

LATS Reflection (Zhou et al., 2024). Cross-trajectory scope with reflection abstraction. After each trajectory completes, an LLM generates a strategy-level summary. Trigger: per-trajectory. In MCTS, reflections can additionally be injected into the reward prompt (denoted "+reward" in Table 2).

LiTS-Fact. Cross-trajectory scope with atomic fact extraction, adapted from mem0 (Chhikara et al., 2025) for the multi-attempt setting. An LLM extracts discrete factual statements from observations. Facts are deduplicated via embedding-based cosine similarity at write time; at inference, all facts from prior attempts are injected into the policy prompt. Trigger: per-step in beam search and MCTS (incremental extraction); per-trajectory in best-of-N (batch extraction from the full trajectory in a single LLM call).

Raw Sibling + LiTS-Fact (ours). Composition of cross-sibling and cross-trajectory scopes (Eq. (5)). Tests whether immediate sibling diversity and accumulated cross-trajectory knowledge are complementary. Only applicable to beam search and MCTS.

H Augmentor Pipeline (Formal)

Each augmentor follows a four-stage pipeline:

1. **Invocation**: an event in the search loop triggers the augmentor (after a step completes, after a trajectory terminates).
2. **Analysis**: $f_k(\mathcal{H}_k)$ produces a context unit c (a textual insight, structured issue, or factual memory).
3. **Persistence decision**: a predicate $g(c) \in \{0, 1\}$ determines whether c is stored in a persistent memory $\mathcal{M}$:

$$\mathcal{M} \leftarrow \mathcal{M} \cup \{c \mid g(c) = 1\}. \tag{7}$$

Raw Sibling context is ephemeral ($g = 0$; exists only during the current expansion). Reflection and fact extraction are persistent ($g = 1$; accumulated across iterations).

4. **Retrieval and injection**: before action sampling, relevant context is retrieved from $\mathcal{M}$ (and any ephemeral context from stage 2) and injected into the policy prompt.

This pipeline unifies all four methods in Table 1 as instances of the same augmentor interface with different choices of scope, abstraction, invocation point, and persistence predicate.

I KGQA Evaluation Subset Selection

For KGQA, tree search methods (Beam Search, MCTS) require a per-step reward model call using Sonnet 4.6 ($3/$15 per 1M input/output tokens), making full 150-example evaluation cost-prohibitive (~$2.90/example $\times$ 150 = ~$435 per configuration, measured from a completed pilot; 5 configurations = ~$2,175 total). To enable direct cross-method comparison within a feasible budget, we evaluate all methods—including best-of-N—on a common 69-example stratified subset.

Subset construction. Starting from the 150-example KGQA dataset (GrailQA subset from AgentBench), we construct the evaluation subset in two steps:

1. **Error examples (60):** All examples where the ReAct baseline (Sonnet 4.6, greedy T=0) produces an incorrect answer (F1 $<$ 1.0). These are the examples where tree search and memory have the opportunity to rescue.

2. **Stratified-correct controls (9):** From the 90 ReAct-correct examples, we draw a stratified random sample of 10 (one per decile of the sorted index range), using fixed seed 42 for reproducibility. One sampled index (idx=1) overlaps with the initial pilot run and is excluded from the subset command, yielding 9 controls. These validate the assumption that search methods do not regress on examples the baseline already solves.

Reproducibility. The subset indices are deterministically generated via stratified random sampling (seed 42) and are provided in our code release. All KGQA configurations in Table 2 — including best-of-N baselines — are evaluated on this identical 69-example set. We report accuracy as the fraction correct within the subset.

J McNemar's Exact Test: Raw P-Values

For every method-pair comparison the paper makes, we report McNemar's two-sided exact binomial test on the discordant pairs. For paired binary outcomes, b rescued (B correct, A wrong) and c regressed (A correct, B wrong) under the null $P(\text{rescue} \mid \text{discordant}) = 0.5$ follow Binomial$(b{+}c, 0.5)$; the exact two-sided p-value is $2 \cdot \sum_{k=0}^{\min(b,c)} \binom{b+c}{k} 0.5^{b+c}$, capped at 1. The exact form is required because several discordant counts in our study fall below the $b{+}c \gtrsim 25$ threshold for the standard chi-square approximation. The full table is reproducible via `paper/lits_memory/results/scripts/mcnemar_test.py`. Significance markers: ** $p{<}0.01$, * $p{<}0.05$, † $p{<}0.10$.

Multiple testing. We do not apply a family-wise correction (e.g., Bonferroni) because several of our headline tests are pre-specified by the unified framework (§3) rather than chosen post-hoc, and because Bonferroni assumes independent tests, a poor fit for our correlated comparisons (the same memory method evaluated on related benchmarks). Benjamini-Hochberg FDR control is more appropriate for our setting; applying it at $q{=}0.05$ to the 23-comparison family above leaves KGQA Beam Raw Sib (uncorrected $p{=}0.003$) at the boundary of the BH threshold, with the three other $p{<}0.05$ findings just above. We discuss this caveat in §6 and report all findings as $p{<}0.05$ uncorrected; readers can re-derive FDR-adjusted p-values from the table above.

K Per-iteration Trace of Pattern 1: Reflection-driven MCTS Rescues

This appendix traces the per-iteration progression of MCTS+Reflection versus MCTS+No-Memory on the two WikiSQL examples that are rescued by MCTS+Reflection but not by Indep+Reflection (§5.1, Pattern 1). Both examples exhibit the same pattern: iteration 0 (reflection-free, since reflection content is generated only after the first simulated trajectory) fails on the same path that MCTS without memory fails on; iteration 1 onwards, after reflection content is available, the search finds a high-reward path. This is consistent with the filtering account in §5.1: the per-step reward model

Table 4: McNemar's exact test results for every method-pair comparison the paper claims, grouped by benchmark. N is the number of examples on which both methods ran. A_{corr} and B_{corr} are the per-method accuracy counts on those N examples; *res* is the count of examples where B is correct and A is wrong (rescues), *reg* the reverse (regressions).

Benchmark	Comparison (A vs. B)	N	A_{corr}	B_{corr}	res	reg	p
WikiSQL	Best-of-N: Refl vs. No Mem	51	25	30	5	2	0.453
	Best-of-N: Fact vs. No Mem	43	20	19	2	3	1.000
	Beam: Raw Sib vs. No Mem	51	14	20	8	2	0.109
	MCTS: Refl vs. No Mem	51	25	31	6	0	0.031^{*}
	MCTS: Fact vs. No Mem	51	25	24	0	1	1.000
	MCTS: Raw Sib vs. No Mem	51	25	24	2	3	1.000
	MCTS+Refl vs. pass@5+Refl	51	27	31	6	2	0.289
WikiTQ	Best-of-N: Refl vs. No Mem	48	19	22	4	1	0.375
	Best-of-N: Fact vs. No Mem	48	19	16	0	3	0.250
	Beam: Raw Sib vs. No Mem	49	13	15	3	1	0.625
	MCTS: Refl vs. No Mem	49	19	22	5	2	0.453
	MCTS: Fact vs. No Mem	49	19	19	2	2	1.000
	MCTS: Raw Sib vs. No Mem	49	19	18	2	3	1.000
	MCTS+Refl vs. pass@5+Refl	49	23	22	1	2	1.000
KGQA (69-subset)	Best-of-N: Refl vs. No Mem	69	22	23	5	4	1.000
	Best-of-N: Fact vs. No Mem	69	22	22	4	4	1.000
	Beam: Raw Sib vs. No Mem	69	5	19	17	3	0.003^{**}
	MCTS: Refl vs. No Mem	69	16	24	10	2	0.039^{*}
	MCTS: Raw Sib vs. No Mem	69	16	24	10	2	0.039^{*}
	MCTS+Refl vs. pass@5+Refl	69	23	24	6	5	1.000
	MCTS+Refl vs. MCTS+Raw Sib	69	24	24	1	1	1.000
Terminal-Bench	Best-of-N: Refl vs. No Mem	89	21	23	8	6	0.791
	Best-of-N: Fact vs. No Mem	89	21	23	5	3	0.727

selects the reflection-induced expansion among the candidate siblings, while pass@N has no analogous selection step within an attempt.

K.1 Case study A: WikiSQL example 26

Question. *"Name the platform for year more than 2006 and developer of 3g studios."* The intended table is named `SWAT Games`; common-sense table names like `game_results` and `Games` exist in the database but do not contain a `Developer` column.

Pass@5+Reflection (Indep+Refl). All five attempts apologise without identifying `SWAT Games`, even though attempts 2–4 see reflections from earlier attempts in their context (the Reflection memory diagnoses *"find a table with platform, year, developer columns"* after attempt 0). Attempts 2 and 3 list tables and inspect schemas of `game_results` / `game score`; both lack `Developer`. The trajectory ends in the apology terminal without ever inspecting `SWAT Games`'s schema.

MCTS+No-Memory. Across iterations 0–4 the best q-mean stays in the 0.68–0.71 range; the best path's last action remains `sql_db_list_tables` or `sql_db_schema(game_results)` in every iteration. The agent never advances to a SELECT against `SWAT Games`.

MCTS+Reflection. Iteration 0 receives no reflection content (the execution log records `ReflectionAugmentor returned empty for traj_key=q/0/...`, since reflection units are generated only after a trajectory completes). It therefore behaves the same way as MCTS+No-Memory: best q-mean is 0.66, best path stops at `sql_db_list_tables`. The first reflection unit is then generated, diagnosing *"wrong tables; need a table with platform, year, developer columns."* At iteration 1, the policy expands a sibling that issues `sql_db_schema(table_names="game_results, SWAT Games")` — the first time `SWAT Games` appears in any candidate. The PRM scores the schema-call branch high; the subsequent SELECT against `SWAT Games` is simulated to terminal at q-mean **0.934**. Iterations 2–4 refine the SQL syntax; the q-mean stays in the 0.91–0.93 range. Table 5 summarises the per-iteration trace.

K.2 Case study B: WikiSQL example 29

Question. *"What is the earliest game with a score of 99-89?"* The intended table is `Game Results` (capitalised, with a space); the look-

Table 5: Per-iteration best q-mean, WikiSQL example 26. MCTS+Refl jumps at iter 1 (first reflection-available iteration); MCTS+No-Mem plateaus.

iter	0	1	2	3	4
MCTS+No Mem	0.68	0.71	0.69	0.69	0.69
MCTS+Refl	0.66	**0.93**	0.92	0.92	0.93

Table 6: Per-iteration best q-mean, WikiSQL example 29. MCTS+Refl's correct path emerges at iter 4 after multiple reflection refinements; MCTS+No-Mem plateaus throughout.

iter	0	1	2	3	4
MCTS+No Mem	0.73	0.70	0.70	0.73	0.70
MCTS+Refl	0.70	0.70	0.73	0.70	**0.89**

alike `game_results` table lacks the required date format.

Trace. The same pattern repeats. MCTS+No-Memory's best q stays in 0.70–0.73 across iterations 0–4; the agent loops between `game_results` and `game score` without finding the correct table. MCTS+Reflection's iteration 0 also fails (q=0.70) on `game_results`; reflection diagnoses *"score format mismatch; check distinct score formats"*; iterations 1–3 refine score-format guesses; iteration 4 reaches `Game Results` through a sibling that the PRM scores high, q-mean reaches **0.89**. The mechanism is the same as case A but with more iterations needed: each subsequent reflection refines the specific format guess, and the PRM filters the non-promising format-guess paths along the way. Table 6 summarises.

K.3 Cross-cell summary and KGQA caveat

The two case studies above show the iter-0 / iter-1+ asymmetry clearly because WikiSQL has a strong floor effect: when the policy picks the wrong table, all five iterations get stuck at the same plateau, so any non-zero progress in MCTS+Reflection is attributable to the reflection content that became available after iteration 0.

KGQA admits four examples that are rescued by MCTS+Reflection but not by Indep+Reflection (idx 3, 37, 46, 110). The per-iteration trace on these examples is less informative for mechanism attribution: KGQA has a denser per-step reward signal (the PRM evaluates each Freebase relation choice in isolation), so iteration 0 q-values already vary substantially across runs. On all four KGQA cases, MCTS+Reflection's iteration 0 best q-mean is within 0.02–0.07 of the final iteration 4 q-mean, while MCTS+No-Memory's iteration 0 q-mean is correspondingly lower than its iteration 4 by a similar margin. The aggregate effect (the $+11.6$pp McNemar-significant gain on the 69-subset) is real, but the per-example mechanism on KGQA is best described as accumulated variance reduction over five MCTS iterations rather than a single reflection-driven rescue moment per example.

We therefore report Pattern 1's mechanism as confirmed at the per-iteration level on WikiSQL (where iter 0 reliably fails) and as aggregate-level only on KGQA. WikiTQ admits no examples in this category and is therefore not analysed at the case-study level.

L Beam Search Give-Up Analysis

On WikiSQL (51 examples) and WikiTQ (49 examples), beam search (beam size 3, Haiku 3.5 policy at T=0.7, Sonnet 4.6 PRM) underperforms greedy ReAct (T=0). Table 7 summarizes the give-up phenomenon.

Table 7: Beam search give-up analysis. *Apol. terminals*: terminals with "I cannot find..." answers. *Sel. apol.*: examples where the best terminal is an apology. *All-apol. states*: expand states where all 3 siblings apologize.

	WikiSQL		**WikiTQ**	
	No Mem	Raw Sib	No Mem	Raw Sib
Accuracy (%)	27.5	**39.2**	26.5	**30.6**
Apol. terminals	90	71	64	53
Sel. apol.	28	23	18	15
All-apol. states	25	—	—	—
Δ (Raw Sib)	+11.7%, -21% apol.		+4.1%, -17% apol.	

Mechanism. At T=0.7 sampling, Haiku 3.5 generates apologize-style continuations in all 3 sibling candidates after encountering SQL errors (e.g., `sqlite_master` syntax error on MySQL). Greedy decoding (T=0) avoids this by argmax selecting a retry token (e.g., "Let me try `information_schema`..."), but the broader sampling distribution at T=0.7 collapses to the apologize mode. Verified on WikiSQL: 0 mixed states (apologize + retry) exist — PRM never faces a retry-vs-apologize choice.

Connection to Chen et al. (2024). Chen et al. show that tree search demands discriminator accuracy $\geq$90% to improve over re-ranking. Our finding identifies a complementary failure mode:

even with a perfect discriminator, tree search cannot help when the *generator* lacks diversity at critical branching points. Cross-sibling memory (Raw Sibling) addresses this by conditioning each candidate on prior siblings' outcomes, expanding the effective generation distribution beyond what temperature-based stochastic sampling achieves alone.

M KGQA Fact Extraction: Regression Case Studies

This appendix examines the per-example regression mechanisms behind LiTS-Fact's accuracy-neutral effect on KGQA (§5.3). The analysis was performed on a 150-example pilot run; the main matrix in Table 2 reports KGQA on the 69-example subset (§4). On the 69-subset, KGQA Indep+LiTS-Fact rescues 4 examples and regresses on 4 (Appendix J); the case studies below illustrate the regression mechanisms, four of which (idx 23, 92, 143, 147) fall within the 69-subset.

Of the 150 KGQA examples, LiTS-Fact regresses on 6 (baseline pass@5 $\rightarrow$ fact fail). All 6 share a common mechanism: facts reduce stochastic diversity by providing a "free" path that the agent follows instead of exploring alternatives. We identify four specific manifestations on KGQA:

Wrong-type path (idx=23). Question: "What Canadian government exists where Baldur von Schirach was born?" (Answer: Constitutional monarchy, type `form_of_government`). Facts record that "canadian" has relation `governmental_jurisdiction.agencies` (type `government_agency`). Agent follows this path $\rightarrow$ wrong answer type. Baseline (4/5 pass) independently discovers `form_of_government`.

Procedural instruction (idx=92, 147). Question: "How many games of Electronic Arts are released in the US?" (Answer: 5). Facts include a procedural instruction: "To find games by a publisher in a region, you must: (1) get games_published, (2) traverse versions, (3) intersect." This indirect path yields 4; the direct relation `game_versions_published` yields 5. Baseline (1–2/5 pass) stochastically finds the direct path.

Wrong-relation lock-in (idx=87). Question: "What programming language paradigm is followed by AngelScript and derivatives of Prolog?" (Answer: Object-oriented programming, single entity). Facts record that "`computer.programming_language.influenced` on Prolog returns the programming languages that Prolog influenced (its derivatives/dialects)." Baseline attempts pick the narrower `computer.programming_language.dialects` relation (returning only Prolog's dialects) and reach the correct single-entity intersection. The fact-augmented attempts consistently use the broader `influenced` relation, returning a superset whose paradigm intersection includes one extra non-answer entity, scoring 0 across all 5 attempts.

Correct-but-irrelevant negative fact (idx=143). Question: "What Facebook founder was influenced by Jeff Bezos?" (Answer: Mark Zuckerberg). Facts state: "Facebook founders do NOT have `influenced_by`." This is correct (the type lacks that relation), but the correct path goes from Jeff Bezos $\rightarrow$ `influenced` $\rightarrow$ people, not from founders $\rightarrow$ `influenced_by`. Agent over-generalizes the negative fact and avoids the influence direction entirely.

Over-exploration (idx=25). Agent finds the correct intermediate answer but facts record deeper relations explored in attempt 0. Subsequent attempts follow these "there's more to explore" signals instead of stopping, hitting max_steps.

N Case Study: Abstraction-Level Conflict

This case study illustrates how combining reflection and fact extraction in the same prompt can degrade both accuracy and efficiency, using WikiSQL Example 0 ("What are the Notes when the Method is decision?").

Setup. Three configurations are compared, all using Haiku 3.5 as policy and Sonnet 4.6 as augmentor LLM, with 5 best-of-N at T=0.9.

Fact extraction alone. After attempt 0 (8 steps, calls `sql_db_list_tables`), the memory backend extracts environmental facts:

> *"The database contains a table named 'Tournament Results'."*
> *"The 'Tournament Results' table has a 'Notes' column."*
> *"The 'Tournament Results' table has a 'Method' column that contains values such as 'decision'."*

Attempts 1–4 skip `sql_db_list_tables` entirely (the agent already knows which table to query) and solve the task in 3–4 steps.

Reflection alone. After attempt 0 fails, the reflection LLM generates:

> *"Step 7 omitted the Notes column from the SELECT clause. Revised Plan: 1. List all tables. 2. Check schema. 3. Run:* `SELECT Notes FROM <table> WHERE Method = 'Decision'`. *4. Return the result."*

Attempts 1–4 follow this plan and achieve higher accuracy (pass@5 = 58.8% across all examples), but still call `sql_db_list_tables` in every attempt because the plan explicitly instructs it.

Fact + Reflection combined. Both augmentors inject into the system prompt simultaneously. The facts say the table is `Tournament Results`; the reflection says "1. List all tables." The agent follows the reflection's explicit step-by-step plan, calling `sql_db_list_tables` in 4/5 attempts despite the facts making this step unnecessary.

Table 8: Example 0: `sql_db_list_tables` calls and step counts across configurations.

	a0	a1	a2	a3	a4
Fact only (list_tables skipped from a1)					
Steps	8	4	3	3	3
list_tables	✓	—	—	—	—
Fact + Reflection (list_tables not skipped)					
Steps	5	7	7	5	6
list_tables	✓	✓	✓	—	✓

Mechanism. The conflict arises because the two memory types operate at different abstraction levels. Facts provide *implicit* environmental knowledge ("table X exists with columns A, B, C") that the agent can use to skip discovery steps — but only if no other signal overrides this. Reflection provides *explicit* procedural plans ("Step 1: list tables, Step 2: check schema") that the agent follows literally. When both are present, the explicit plan dominates: the agent treats the reflection as an instruction sequence rather than consulting the facts to determine which steps are already unnecessary.

Aggregate effect. Across all 51 WikiSQL examples, the `list_tables` skip rate drops from 77% (fact only) to 20% (fact + reflection) in attempts 1–4. Pass@5 drops from 58.8% (reflection only) to 52.9% (fact + reflection), suggesting that the added noise from conflicting signals also hurts accuracy. The same pattern holds on WikiTQ (70% → 23% skip rate; 49.0% → 46.9% pass@5).

O Retrieval Policies for Cross-Trajectory Fact Memory

We expand on the argument in §6 that selective retrieval cannot resolve the diversity-vs-efficiency tradeoff. Table 9 enumerates four retrieval policies along the dimensions of diversity preservation and efficiency (skipping redundant discovery). Only *Inject All* (LiTS-Fact in our experiments) and *None* (baseline) are evaluated in this paper; the other two are included as a design-space analysis, not empirical results.

Why no policy resolves the tradeoff. Diversity loss is triggered by the *presence* of any positive prior facts in the prompt, not by which subset is chosen — once injected, the LLM treats them as ground truth and biases toward consistency. Conversely, the efficiency benefit of fact memory comes from making environmental knowledge available across trajectories — which similarity-based retrieval undermines whenever the relevant fact's discovery context differs from the current step's context. The Pareto frontier in this design space is therefore narrow, and our *Inject All* configuration sits on the high-efficiency, low-diversity end of it. Mitigations identified in §6 (negative-fact extraction, candidate-vs-truth framing) operate on *what is stored* or *how it is presented*, not on the retrieval axis.

P Efficiency Metrics

Q Compute Cost

Table 11 reports the API cost and token usage of each configuration. All costs are computed from logged token counts at Bedrock pricing: Haiku 3.5 ($0.80/$4.00 per 1M input/output tokens) for WikiSQL/WikiTQ/Terminal-Bench policy; Sonnet 4.6 ($3.00/$15.00) for KGQA policy, all per-step reward models, and all memory/augmentor LLMs. Raw Sibling does not invoke a separate LLM (it injects raw observations directly), so its cost equals the No-Memory baseline. The total experiment cost is approximately **$1,384**.

Table 9: Retrieval policies for cross-trajectory fact memory. Only *Inject All* and *None* correspond to evaluated configurations (LiTS-Fact and No Memory respectively in Table 2); *Similar* and *Dissimilar* are predicted behaviors. **Diversity**: ability of subsequent attempts to explore alternative paths. **Efficiency**: ability to skip redundant environmental discovery.

Policy	What is injected	Diversity	Efficiency
Inject All (LiTS-Fact)	Union of all prior trajectories' facts	Reduced — agent anchors on the most-supported prior path	High — known facts let the agent skip redundant discovery
Similar to current state (predicted)	Facts most embedding-similar to the current step	Further reduced — retrieval surfaces precisely the prior path	Inconsistent — relevant fact retrieved only when current step is surface-similar
Dissimilar to current state (predicted)	Facts least similar to the current step	Could push exploration; could also distract	Low — facts that enable shortcuts are systematically excluded
None (No Memory baseline)	—	Maximal — attempts are i.i.d. samples from the policy	Low — every attempt re-discovers environmental structure

Table 10: Efficiency metrics for best-of-N. *Steps*: mean trajectory length (lower is better). *Skip*: fraction of attempts 1–4 that skip the initial discovery call (higher = more efficient). For SQL: skipping `list_tables`; for KG: not applicable. *Cost*: policy cost / total cost per 5-attempt run. Total includes augmentor LLM (Sonnet \$3/\$15). SQL policy: Haiku \$0.80/\$4.00; KG policy: Sonnet \$3/\$15.

		Steps	Skip %	Cost (pol. / tot.)
WikiSQL	No Memory	6.1	4	2.20 / 2.20
	Reflection	6.4	5	3.26 / 5.02
	LiTS-Fact	**4.9**	**77**	1.68 / 2.66
	Fact + Refl.	5.9	20	3.54 / 5.89
WikiTQ	No Memory	5.8	5	1.89 / 1.89
	Reflection	5.9	8	2.71 / 4.35
	LiTS-Fact	**4.7**	**70**	1.50 / 2.34
	Fact + Refl.	5.6	23	2.75 / 5.28
KGQA	No Memory	8.9	—	60 / 60
	Reflection	8.9	—	76 / 82
	LiTS-Fact	**6.6**	—	58 / 64

Table 11: Per-configuration API cost and token usage. N = evaluated examples. *Pol.* = policy LLM calls/-tokens; *Sup.* = supervisor (PRM + memory/augmentor) calls/tokens. KGQA Indep runs on 150 examples; Beam/MCTS on 69-subset. †Not evaluated (see Table 2).

			Policy		Supervisor		
Bench.	Config	N	calls	tok (M)	calls	tok (M)	Cost
WikiSQL	Indep No-Mem	51	1310	2.2	—	—	\$2
	Indep Refl	51	1379	3.5	255	0.2	\$5
	Indep Fact	51	985	1.6	255	0.2	\$3
	Beam No-Mem	51	723	1.2	714	1.1	\$7
	Beam Raw Sib	51	729	1.2	724	1.1	\$7
	MCTS No-Mem	51	2880	5.4	2879	5.1	\$31
	MCTS Refl	51	2910	7.3	2910	5.1	\$31
	MCTS Raw Sib	51	2871	5.4	2870	5.1	\$31
	MCTS Fact	51	3027	6.1	6054	6.4	\$40
WikiTQ	Indep No-Mem	49	1183	1.9	—	—	\$2
	Indep Refl	49	1197	2.9	245	0.2	\$4
	Indep Fact	49	909	1.5	245	0.1	\$2
	MCTS No-Mem	49	2619	4.8	2619	4.5	\$27
	MCTS Refl	49	2682	6.7	2900	4.8	\$31
	MCTS Raw Sib	49	2454	4.5	2454	4.2	\$26
KGQA	Indep No-Mem	150	6698	16.9	—	—	\$60
	Indep Refl	150	6234	21.5	750	0.7	\$82
	Indep Fact	150	4947	15.6	750	1.5	\$64
	Beam No-Mem	69	2202	6.0	2202	3.9	\$39
	Beam Raw Sib	69	1794	5.3	1794	3.4	\$36
	MCTS No-Mem	69	9043	26.9	9042	18.9	\$186
	MCTS Refl	69	7953	28.9	8298	16.5	\$183
	MCTS Raw Sib	69	8163	26.9	8163	18.0	\$188
T-B	Indep No-Mem	89	6675	96.4	—	—	\$85
	Indep Refl	89	6952	117.8	369	2.4	\$112
	Indep Fact	89	6461	110.4	440	2.4	\$105