VETEX: BEYOND MORE EXPERIENCE, TOWARD FILLING EXPERIENCE GAPS IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training-free experience reuse typically improves frozen LLM agents through two routes: extracting reusable corrective experience from failed and successful trajectories, or using successful trajectories, directly or after compaction, as few-shot references. However, both routes lack a principled criterion for deciding which experience should actually be admitted for reuse. Reorganizing the receiver's own failure history alone cannot supply correct behaviors missing from that history; similarly, leaving a successful trajectory unprocessed can preserve misleading actions or unsupported state assumptions, while compaction can remove critical details needed by later decisions. We propose VetEx, which combines corrective experience construction targeting the receiver's behavioral gaps with reference trajectory cleaning. For corrective experience, VetEx anchors same-task success–failure contrasts on receiver failures and admits donor success trajectories only for tasks the receiver has failed but never succeeded on, filling gaps in positive behavioral evidence. For task references, VetEx cleans retrieved successful trajectories, reducing redundant content while preserving task-relevant action–observation relations, and injects them together with abstract corrective experience into the frozen agent's initial context. Across ALFWorld, HotpotQA, and WebShop with three frozen receiver models, VetEx outperforms ReAct in all nine settings and achieves a higher three-run mean than the strongest reproduced experience-learning baseline selected by single-run performance in each setting, by an average three-run mean margin of 5.12 points. Controlled analyses show that gains from experience extraction depend on whether newly added evidence fills behavioral gaps in the receiver's history rather than on experience quantity alone, and that corrective experience and cleaned few-shot trajectories are complementary.

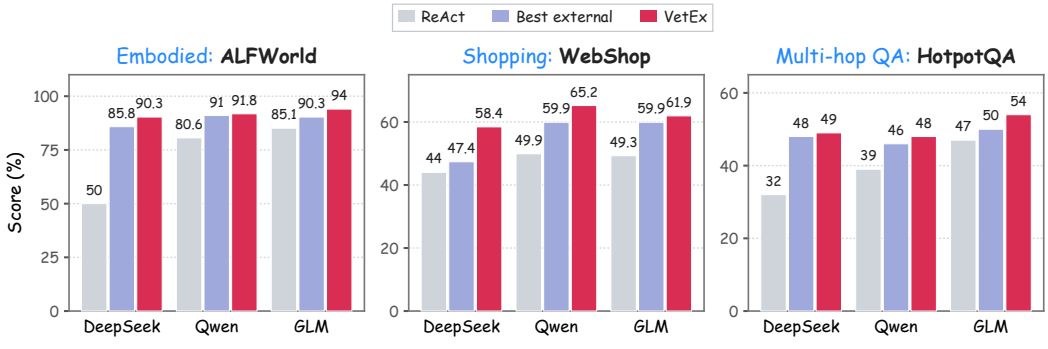


Figure 1: Headline results across nine benchmark–receiver settings. Each group compares ReAct, the best external baseline, and VetEx. Best external is selected independently per setting from ExpeL, ReasoningBank, Synapse, AdaMEM, and SkillGen; full scores and metric definitions appear in Table 2. Panels use different zero-based vertical scales.

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.

1 INTRODUCTION

LLM agents solve interactive tasks through repeated cycles of reasoning, action, and observation. ReAct interleaves language reasoning with environment actions; SayCan grounds high-level plans in executable skills; Toolformer and WebGPT show how agents can use external tools for retrieval, computation, and browsing; and Voyager demonstrates that agents can accumulate reusable programs over extended task streams (Yao et al., 2023; Ahn et al., 2022; Schick et al., 2023; Nakano et al., 2022; Wang et al., 2023). Recent surveys place these capabilities within broader frameworks for agentic reasoning, tool use, and interactive evaluation (Wei et al., 2026; Liu et al., 2024b). These systems show that performance depends not only on the underlying language model, but also on how an agent uses feedback, tools, and prior interaction experience.

Each execution leaves a trajectory containing model decisions, environment feedback, and the final outcome. Successful trajectories record effective behavior, whereas failures reveal decisions that require correction. Training-free experience learning reuses these records without updating the downstream model parameters, forming part of a broader view of external memory and agent cognition (Sumers et al., 2024; Zhang et al., 2025; Wang et al., 2025). Existing methods mainly reuse history in two forms: abstract corrective experience that can transfer across tasks, and successful trajectories that provide task-level references at the start of a new task (Zhao et al., 2024; Ouyang et al., 2026; Zheng et al., 2024; Zhang et al., 2026).

The first route has a gap in its coverage of positive evidence. A failure trajectory is easier to convert into a reliable corrective rule when a successful behavioral reference is available for the same task. If the receiver has failed on a task but has no successful run for that task, reflection over the failure and transfer from successful runs on other tasks may still help, but without the specific positive reference they are less likely to identify a reliable corrective strategy for that task. Even sophisticated extraction and aggregation procedures may not fully remove this limitation (Fu et al., 2024a; Gupta et al., 2024; Gao et al., 2025; Ma et al., 2026; Feng et al., 2026).

The second route has a gap in how it prepares task references. A successful trajectory is not necessarily a suitable demonstration for a new receiver. Raw trajectories may contain repeated actions, irrelevant detours, or decisions unsupported by preceding observations. Direct injection will provide the receiver with these problematic behaviors as task-solving references, motivating trajectory cleaning before injection. However, deleting steps during cleaning may also remove observations or intermediate states needed for later decisions (Shi et al., 2026; Hsu et al., 2026; Yu et al., 2026; Kang et al., 2026; Jiang et al., 2023; Kagaya et al., 2024).

Figure 2 summarizes these two routes and the gaps that motivate our design. VetEx addresses them with two independent offline pipelines. For corrective experience, it anchors each item on a receiver

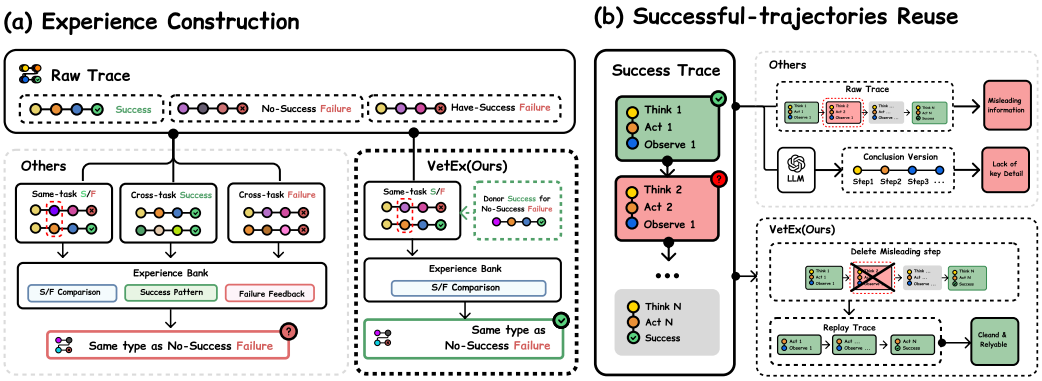


Figure 2: Two routes for training-free experience reuse and their limitations. (a) Complex reflection, extraction, or aggregation over a receiver's failure history cannot recover correct behavior absent from that history; constructing corrective experience therefore requires positive behavioral evidence, which VetEx supplements with donor success records only for tasks the receiver has failed but never solved. (b) Successful trajectory reuse provides concrete task references, but raw traces may contain redundant or observation-unsupported decisions; VetEx cleans the retrieved trace while preserving the original action–observation dependencies.

failure and forms a uniform contrast between successful and failed trajectories on the same task. A donor success record is admitted only when the receiver has no observed success for that task to expand positive behavioral coverage. Self and Peer pairs are compiled separately and reconciled under the same experience schema. For task references, VetEx retrieves successful trajectories similar to the current task, then removes redundant or unsupported behavior while preserving the original order and source actions. The compact reference will be used only when it passes the native replay; otherwise, it uses the original trace. The resulting corrective experience and cleaned task references are concatenated into the frozen receiver's initial context before its first action.

The headline results in Figure 1 show the overall effect of these design choices. Across three benchmarks and three frozen receiver models, VetEx outperforms both ReAct and the strongest external baseline in all nine benchmark–receiver settings. Three-run evaluations further confirm a higher mean score than the selected external comparator in every setting, with an average margin of 5.12 points. These results suggest that effective experience reuse depends on supplying behaviorally useful evidence and task-relevant references, rather than simply increasing the amount of stored experience.

Our work makes three key contributions: (1) We distinguish gaps in positive evidence coverage from experience organization and examine their downstream utility separately; (2) We propose **VetEx**, a training-free framework with two independent offline pipelines: same-task success–failure contrasts for corrective experience, and reference trajectory cleaning with environment replay validation for executable references; (3) We evaluate VetEx across nine benchmark–receiver settings and task-level behavioral cases, quantifying the utility and operating boundaries of these two experience types.

2 Related Work

Language agents and experience memory. LLM agents use reasoning, environment interaction, executable skills, and external tools to solve interactive tasks (Yao et al., 2023; Ahn et al., 2022; Schick et al., 2023; Nakano et al., 2022). Their interaction histories can be retained as external memory to support subsequent tasks, skill accumulation, and continual adaptation (Park et al., 2023; Wang et al., 2023; Cai et al., 2025), within broader studies of agent architectures and memory systems (Sumers et al., 2024; Zhang et al., 2025; Wei et al., 2026; Liu et al., 2024b). We focus on training-free, offline experience reuse for frozen receivers, and distinguish two forms of reuse: constructing corrective experience and directly reusing successful trajectories as task references.

Corrective experience: evidence and organization. Existing methods construct reusable guidance from demonstrations, offline interactions, pre-collected experience, or experience shared across agents (Zhao et al., 2024; Fu et al., 2024b; Chen et al., 2024; Liu et al., 2025; Qian et al., 2024). They further organize this evidence through reflection, experience extraction, structured memories or skills, and memory consolidation (Shinn et al., 2023; Madaan et al., 2023; Fu et al., 2024a; Gupta et al., 2024; Gao et al., 2025; Ouyang et al., 2026; Ye et al., 2026; Shalev et al., 2026). AutoGuide extracts guidelines from same-task trajectories with different returns, while ExpeL combines local success–failure contrasts with cross-task insights (Fu et al., 2024b; Zhao et al., 2024). These methods primarily determine how available evidence is extracted and organized. VetEx instead asks whether the receiver has the positive same-task evidence required for such contrasts in the first place. When the receiver has failed a task but has no successful record, VetEx admits a same-task donor success record to fill this evidence gap while retaining a uniform local failure–success construction unit.

Reusing successful trajectories: form and preparation. Successful trajectories can be reused as demonstrations, compressed memories, or state-indexed references (Zheng et al., 2024; Zhao et al., 2024; Zhang et al., 2026), with prior work studying retrieval, compression, memory organization, and context placement (Shi et al., 2026; Hsu et al., 2026; Yu et al., 2026; Kang et al., 2026; Jiang et al., 2023; Liu et al., 2024a; Kagaya et al., 2024). ReasoningBank uses an LLM to judge trajectory outcomes for memory extraction, while Learn-by-Interact removes duplicate action–observation pairs and evaluates synthesized instruction–trajectory pairs (Ouyang et al., 2026; Su et al., 2025). VetEx focuses on a different question: whether a retrieved successful trajectory remains a reliable task reference after preparation. It removes redundant or observation-unsupported behavior while preserving task-relevant action–observation dependencies, validates the retained action sequence through native replay, and falls back to the raw trajectory when validation fails.

3 VETEX: GAP-GATED EXPERIENCE AND TASK-REFERENCE TRAJECTORY CLEANING

3.1 OVERVIEW OF VETEX

VetEx uses historical trajectories to construct cross-task corrective experience and task-level references matched to the current task, and injects both into the receiver's initial context at task start. Figure 3 presents the overall VetEx architecture: the offline stage separately constructs corrective experience and task-reference trajectories, which are then jointly injected into the frozen receiver before task execution. The following sections describe the construction of corrective experience and reference trajectory preparation and injection. The full algorithmic details are provided in Appendix B.1.

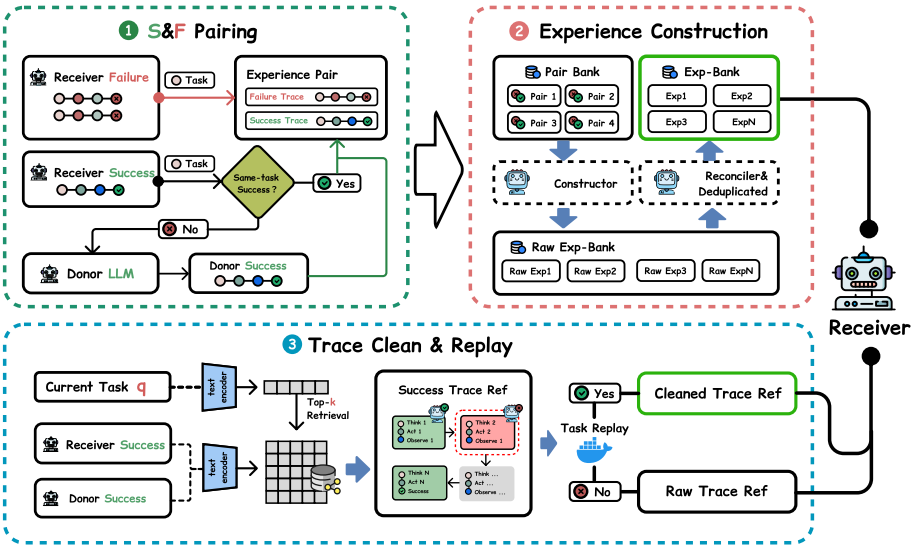


Figure 3: Overview of VetEx. Offline compilation constructs receiver-conditioned corrective experience from same-task success–failure contrasts, while reference trajectory cleaning provides behavioral references. Both are injected into the frozen receiver model at task start.

3.2 RECEIVER-CONDITIONED CORRECTIVE EXPERIENCE CONSTRUCTION

We consider a frozen receiver model r that must solve subsequent tasks using interaction trajectories from past tasks. Each historical record contains a task identity, a visible action–observation sequence, the task outcome, and the model that executed the trajectory. Let $\mathcal{D}_r^+$ and $\mathcal{D}_r^-$ denote the receiver's successful and failed trajectories, respectively, and let $\mathcal{D}_p^+$ denote successful trajectories from an optional donor model p.

VetEx estimates task-level capability gaps from the outcomes of multiple historical rollouts at different decoding temperatures. A task is treated as having an observed capability gap when the receiver has no observed success across the available attempts. Let $t(d)$ denote the task identity associated with trajectory d, and let $\mathcal{U}_r$ denote the tasks for which the receiver has no observed success across the available attempts. The Self channel pairs receiver failure records with same-task receiver success records; the Peer channel uses a same-task donor success record only for tasks in $\mathcal{U}_r$:

$$\mathcal{P}_{\text{self}} = \{(d^-, d^+) : d^- \in \mathcal{D}_r^-,\ d^+ \in \mathcal{D}_r^+,\ t(d^-) = t(d^+)\},$$
$$\mathcal{P}_{\text{peer}} = \{(d^-, d^+) : d^- \in \mathcal{D}_r^-,\ d^+ \in \mathcal{D}_p^+,\ t(d^-) = t(d^+),\ t(d^-) \in \mathcal{U}_r\}.$$

A frozen compiler compiles each complete failure–success pair into a compact corrective experience item. Each item specifies its applicability condition, the behavior to take or avoid, and links to the corresponding success and failure trajectories. Self and Peer are compiled separately; duplicate expressions are merged and conflicts reconciled within each channel before the outputs are combined into the receiver's All bank G_r^{all}. Detailed implementation details are provided in Appendix B.2.

3.3 Task-Start Trace Preparation and Injection

Given a new task q, VetEx retrieves the Top-K most similar candidates from the successful-trajectory bank built from complementary folds, with $K = 2$ in our experiments. VetEx processes only candidates that may enter the current context instead of pre-compressing the full trajectory bank.

For each retrieved trajectory, a benchmark adapter first parses its native actions and observations. The LLM-driven trajectory selector preserves the original order while removing actions unsupported by preceding observations, repeated or environment-rejected behavior, and detours that do not contribute to the subsequent successful path. The cleaning prompt and reference-trajectory formats are given in Appendix B.3.

After selection, the adapter replays the selected original actions in the source task environment without the actor, using the benchmark's native action format. We use the compact view only if every selected action executes and the replay reaches the benchmark's official terminal success criterion; otherwise, we retain the raw source trajectory.

Let V_q denote the final reference views of the Top-K candidates produced by the preparation procedure above. Before the receiver's first action, its initial context is

$$P_q = \mathrm{Concat}\left(P_0, G_r^{\mathrm{all}}, V_q\right).$$

Here, P_0 is the benchmark's base prompt, G_r^{all} is the All bank and supplies cross-task corrective experience, and V_q supplies task-matched reference trajectories. The three components are concatenated in the displayed order to form the receiver's initial context. The trajectory views occupy a fixed number of demonstration slots. After this task-start injection, the receiver executes through the benchmark's native action–observation interface without online modification of the experience items or trajectory views.

4 Experiments

4.1 Experimental Setup

We evaluate VetEx on ALFWorld, HotpotQA, and WebShop with DeepSeek-V3.2, Qwen3.6-27B, and GLM-5.1 as receivers, forming a 3×3 benchmark–receiver matrix (Shridhar et al., 2021; Yang et al., 2018; Yao et al., 2022). We compare with ReAct, ExpeL, ReasoningBank, Synapse, AdaMEM, and SkillGen under matched tasks, action interfaces, and environment budgets (Yao et al., 2023; Zhao et al., 2024; Ouyang et al., 2026; Zheng et al., 2024; Zhang et al., 2026; Ma et al., 2026).

Table 1: Audited experience-use configurations and estimated context tokens. rb represents ReasoningBank.

Method	Donor trajectory	Experience extraction	Trajectory reference	Consumed traces	Context tokens (k)
ReAct	×	×	×	0	–
ExpeL	×	✓	✓	2,651	1.68
rb	×	✓	×	6,606	0.40
Synapse	×	×	✓	2,736	1.00
AdaMEM	×	✓	×	2,575	2.05
SkillGen	×	✓	×	878	0.67
VetEx	✓	✓	✓	**1,028**	**3.07**

To prevent evaluation leakage, we use fold-rotated evaluation. ALFWorld contains 134 tasks from the validation-unseen split and is divided into two folds of 67 tasks. For each evaluation fold, the remaining 67 tasks form the experience construction pool. HotpotQA and WebShop each use a fixed set of 100 tasks, partitioned into four folds of 25 tasks. For each held-out fold, the remaining 75 tasks form the experience construction pool. Rotating the held-out fold across the complete task set ensures that every task is evaluated once and that each reported benchmark aggregate covers the full task set. Detailed experimental configurations and reproduction procedures are provided in Appendix A.

Table 1 summarizes how each method constructs and uses experience, together with its trajectory usage during offline experience construction and the injection of experience tokens during task execution. Consumed traces reports the total number of trajectories used in the offline experience-construction stage. Context tokens reports the estimated mean number of experience tokens injected into the agent's context during task execution, in thousands; the reported value averages the three benchmark-level means. Detailed accounting procedures and values are provided in Appendix E.

4.2 Main Results

Table 2: Headline results across three benchmarks and three receiver models. ALFWorld, HotpotQA, and WebShop report success rate (%), EM (%), and 100× mean reward, respectively. Small signed values show score differences from ReAct in the same setting. Bold and underlined scores denote the best and second-best results in each row, including ties. VetEx-E uses All Experience only; VetEx-Exp+RT additionally injects raw task-start reference trajectories. The full VetEx column is shaded blue.

Benchmark	Control	External baselines					VetEx variants & full method		
	ReAct	ExpeL	ReasoningBank	Synapse	AdaMEM	SkillGen	VetEx-E	VetEx-Exp+RT	**VetEx (Ours)**
DeepSeek-V3.2									
ALFWorld	50.0	85.8 +35.8	49.3 −0.7	49.3 −0.7	76.9 +26.9	68.7 +18.7	76.9 +26.9	84.3 +34.3	**90.3** +40.3
HotpotQA	32.0	48.0 +16.0	45.0 +13.0	26.0 −6.0	27.0 −5.0	41.0 +9.0	**49.0** +17.0	43.0 +11.0	**49.0** +17.0
WebShop	44.0	46.0 +2.0	47.4 +3.4	44.4 +0.4	40.0 −4.0	22.4 −21.6	52.8 +8.8	52.5 +8.5	**58.4** +14.4
Qwen3.6-27B									
ALFWorld	80.6	91.0 +10.4	76.9 −3.7	79.9 −0.7	73.1 −7.5	78.4 −2.2	**91.8** +11.2	91.0 +10.4	**91.8** +11.2
HotpotQA	39.0	41.0 +2.0	44.0 +5.0	43.0 +4.0	46.0 +7.0	43.0 +4.0	46.0 +7.0	46.0 +7.0	**48.0** +9.0
WebShop	49.9	53.6 +3.7	52.5 +2.6	59.9 +10.0	54.1 +4.2	44.2 −5.7	59.0 +9.1	60.4 +10.5	**65.2** +15.3
GLM-5.1									
ALFWorld	85.1	90.3 +5.2	85.1 +0.0	83.6 −1.5	74.6 −10.5	88.1 +3.0	90.3 +5.2	**95.5** +10.4	94.0 +8.9
HotpotQA	47.0	46.0 −1.0	47.0 +0.0	47.0 +0.0	50.0 +3.0	49.0 +2.0	53.0 +6.0	52.0 +5.0	**54.0** +7.0
WebShop	49.3	59.9 +10.6	51.2 +1.9	48.5 −0.8	58.0 +8.7	45.1 −4.2	58.7 +9.4	58.6 +9.3	**61.9** +12.6

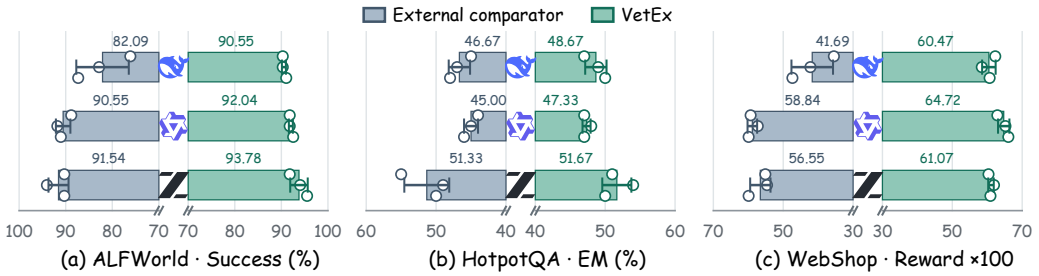


Figure 4: Three-run score comparisons between the external comparator selected in Table 2 (left) and VetEx (right). Rows correspond to DeepSeek, Qwen, and GLM; open circles show the three individual runs, and whiskers show the mean ± sample SD. Each benchmark uses its own truncated score scale.

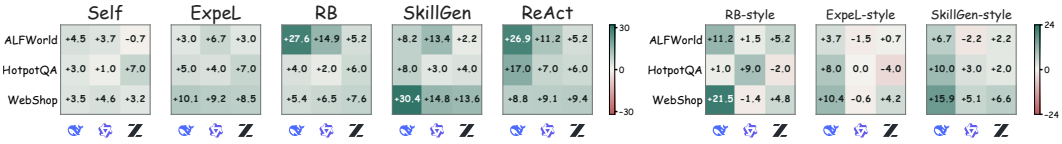
As shown in Table 2, VetEx exceeds ReAct and the strongest external baseline in all nine settings. Averaged across settings, it improves over ReAct by **15.08** percentage points, with a maximum gain of 40.3 points on ALFWorld–DeepSeek. Against the strongest external baseline in each setting, the average gain is 3.81 points, reaching **11.0** points on WebShop–DeepSeek.

The advantage is more pronounced on ALFWorld and WebShop, which require multi-step environment interaction. Average gains over the strongest external baselines are 3.00 and 6.14 points, respectively, with improvements on all three receiver models. These results demonstrate that VetEx improves performance on complex tasks requiring sequential decisions and environment feedback.

For each setting, we select the strongest external baseline by its score in Table 2 and report three repeated evaluations of that method. In Figure 4, VetEx achieves a higher mean score than the listed external comparator in all nine settings, with an average margin of 5.12 points. The average gains on ALFWorld and WebShop are 4.06 and 9.73 points. Moreover, VetEx exhibits lower sample standard deviations in seven of the nine settings, indicating generally more stable performance across repeated runs. Complete repeated-evaluation results and standard deviations are reported in Appendix C.1.

4.3 Corrective Experience Construction

In this section, we examine how positive-evidence coverage, coverage gating, and experience granularity affect the downstream utility of corrective experience. All comparisons in this section use experience-only inference to exclude the influence of task-start reference trajectories.



(a) Experience sources and methods (b) Experience organization

Figure 5: Experience ablations. In the figure, Self denotes Self Experience, ExpeL denotes ExpeL insight-only, and RB denotes ReasoningBank; RB-style, ExpeL-style, and SkillGen-style denote the corresponding organization variants. (a) All Experience minus each named control; positive values favor All. (b) With source material fixed, uniform same-task local contrast minus each named organization; positive values favor local contrast. Negative values favor the corresponding control.

Source expansion helps when it fills positive-evidence gaps. Figure 5(a) uses the same All Experience evaluation throughout. All improves over Self Experience in eight of the nine settings and declines in one. The separate paired source audit in Appendix C.2 provides the behavioral evidence. For Qwen–WebShop, the number of same-task failure–success pairs increases from 81 to 117, yielding 14 new corrective themes. Task *fixed_63* illustrates the contribution of a newly covered behavior. Self associates the product attribute *kahuna* with an unverified brand and exhausts its budget on incorrect queries. The added Peer Experience contains a rule absent from the Self bank: do not infer a brand from an attribute name and add that guess to the query. With All Experience, the Actor maintains attribute-focused search, inspects a candidate page, selects *kahuna* and *20 plus*, and completes the purchase, increasing reward from 0 to 1.0. This rule originates from a receiver-failure–donor-success pair on a different task. Appendix C.4.1 provides a detailed account of this example, including its source lineage and target-rollout comparison.

Source expansion can also cause regressions. Figure 5(a) shows a -0.7-point change relative to Self on GLM–ALFWorld. In the separate paired source audit, adding 54 failure–success pairs and 13 experience entries leaves the number of successful tasks unchanged at 122/134. In a representative task, the added experience does not cover the required distinction between object types: both Self and All treat a *handtowel* as the requested *cloth*, leaving the original error unresolved. In the separate matched source-expansion audit reported in Appendix C.2, Qwen–HotpotQA shows four repairs and eight regressions, giving a paired net change of -4.0 EM points. In four of these regressions, the Actor has already identified the correct answer in its reasoning but continues searching or querying until the seven-step budget is exhausted. These results show that pair counts and bank size alone do not determine the value of source expansion. Its utility also depends on whether the added experience addresses the relevant failure and whether it induces unnecessary subsequent actions. Appendices C.4.2 and C.4.3 provide detailed accounts of the GLM–ALFWorld null case and Qwen–HotpotQA regression.

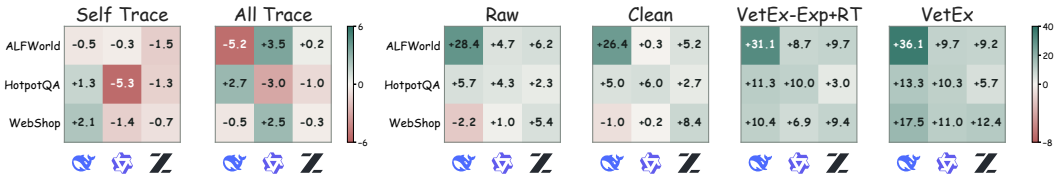
Holding source material, the experience extraction model, final reconciliation, and the inference protocol fixed, we vary only the experience-construction unit before reconciliation, comparing uniform local contrasts with ReasoningBank-style, ExpeL-style, and SkillGen-style organization. Across all three benchmarks, Figure 5(b) shows that uniform local contrasts achieve the highest score in six of the nine Actor–benchmark settings, including one tie; the absolute scores are reported in Table 11. The alternative organization styles win only in isolated settings, whereas uniform local contrasts show the more stable cross-setting performance. Thus, under the same source coverage, mixing evidence with different granularities and roles before extraction does not consistently improve downstream experience utility; uniform same-task local contrasts are the more stable choice across settings.

Coverage gating reduces extraction cost. Table 12 compares Gated All with Ungated All. Gated All admits the donor's success record for Peer construction only when the receiver has a failure record but no success record for that task. Ungated All retains the Self channel while allowing Peer pairs also for failed tasks with receiver success records. The Token usage row of each benchmark

in Table 12 reports the reduction in estimated input-evidence tokens for Peer Experience extraction under the same source pool. The arithmetic mean of the nine setting-specific reductions is **48.52%**. In downstream evaluation, Gated All scores higher in six of the nine settings. These results show that expanding Peer admission does not consistently improve performance, whereas gating by gaps in successful evidence reduces experience-extraction input overhead. The complete score, Peer-pair, and token-usage rows are reported in Table 12 in Appendix C.6.

4.4 Trajectory Sources and Cleaning

We next examine how trajectory-bank expansion and trajectory cleaning affect downstream performance. Execution records provide further evidence about their effects and boundaries.



(a) Trajectory source expansion (b) Trace preparation and joint use

Figure 6: Trajectory ablations: differences in three-run mean scores. In panel (a), the left matrix reports All Trace (the source-expanded raw trace bank) minus Self Trace (the self-only raw trace bank), with no corrective experience supplied to either condition. The right matrix reports All Trace with the same All Experience supplied minus Self Trace with the same All Experience supplied. Raw and Clean denote the corresponding trace preparations. In panel (b), each configuration is compared with ReAct: positive values favor the named configuration and negative values favor ReAct.

Donor trace expansion is not consistently beneficial. Starting from a bank containing only the receiver's own successful trajectories, we additionally admit donor successful trajectories while keeping the retrieval method and Raw Trace injection protocol unchanged. As shown in Figure 6(a), when only trajectories are injected, adding donor successful trajectories improves two of the nine settings and lowers seven, with an average change of -0.85 points. When both conditions additionally receive the same corrective experience, four settings improve and five decline, averaging -0.12 points. With corrective experience also injected, Qwen gains 3.48 and 2.53 points on ALFWorld and WebShop, respectively, but loses 3.00 points on HotpotQA; DeepSeek loses 5.23 points on ALFWorld. Thus, adding donor successful trajectories as retrievable references is not by itself sufficient to improve downstream task performance.

Cleaning is most effective with All Experience. Based on mean scores over three independent runs, Clean Trace-only improves over Raw Trace-only in four of the nine settings, with an overall nine-setting mean change of -0.31 points. With All Experience fixed, VetEx improves over VetEx-Exp+RT in eight settings, averaging $+2.73$ points; the gain reaches 7.09 points on WebShop–DeepSeek (Figure 6(b); Table 6). The component ALFWorld Book–DeskLamp case provides a detailed illustration of this mechanism in task execution: holding All Experience fixed, adding Clean Trace provides a task-level reference for locating the book, taking it, returning to the observed lamp, and using the lamp. The Experience-only Actor exhausts its budget after returning to the desk, whereas the combined configuration completes the final lamp-use action. In the HotpotQA W. H. Shipman House case, holding the source trajectories and compact view fixed, Clean Trace-only continues searching after the available evidence supports the answer and never submits it. With All Experience added, the Actor preserves the evidence, performs a targeted verification, and executes *Finish[Hawaii County]*. The former shows how Trace supplies concrete execution steps; the latter shows how Experience constrains verification and termination when using an existing reference. Complete case trajectories are provided in Appendix D.2.1.

At the same time, we find that using cleaned trajectories alone does not yield consistent performance gains; its utility depends on which corrective behavior the receiving task still needs. Two HotpotQA tasks use the same two source trajectories in the same order, and the cleaning operation changes only one of the two source trajectories. Both tasks are evaluated without corrective experience. On Alice David, both Actors initially follow a productive search path toward the correct answer. The

cleaned reference presents a concise behavior pattern, helping the Clean Actor quickly recover the relevant evidence, recognize that the available evidence already supports the answer, and submit it successfully. The uncleaned reference instead leads to a longer sequence of exploratory, divergent, and repeated searches, and the Raw Actor ultimately does not submit an answer. On Celene, the situation is reversed. Both Actors initially enter a search path that is off target. In this state, the exploratory and divergent behavior retained in the raw reference provides a useful example of how to move beyond an off-target search. The Clean Actor lacks this experience and instead follows an incorrect cue from the question, eventually leaving no remaining step to submit the answer. These cases show that the same exploratory behavior can encourage unnecessary searching when the Actor is already close to the answer, but provide a useful reference for changing direction when the Actor has moved off target. Consequently, the same cleaning operation can produce different effects across tasks and receiving Actors. Details of the above cases are provided in Appendix D.2.2.

4.5 Relations between Capability and Quality

We examine whether the capability of models involved in experience construction is related to the quality of the resulting experience. We replace only the experience extraction model (compiler) responsible for pairwise extraction and final reconciliation, comparing GLM-5.1, Qwen3.6-27B, and GPT-5.6-sol (high). Relative to GLM-5.1, extracting experience with GPT-5.6-sol (high) raises the DeepSeek receiver's score by 1.56 points but lowers the Qwen and GLM receivers' scores by 8.10 and 5.82 points, respectively. Full three-run results in Table 15. These results show that a stronger experience extraction model does not necessarily produce experience with greater downstream utility.

Then, we replace the experience donor for the GLM receiver from Qwen to GPT-5.6-sol (high). Relative to the Qwen donor, the GPT-5.6-sol (high) donor obtains higher scores in three of the four configurations: VetEx-Exp, VetEx-Exp+RT, and the full VetEx configuration improve by 1.46, 3.44, and 0.87 reward $\times 100$ points, respectively, whereas Clean Trace-only decreases by 1.94 points (Table 16). These gains are insufficient to reflect the substantial difference in aggregate capability between the two donor models, indicating that donor capability and the downstream utility of its successful trajectories are not simply positively correlated. This result provides indirect support for our earlier conclusion that source expansion helps when it fills positive-evidence gaps.

4.6 Limitations and Operating Boundaries

Trajectory cleaning may remove decision information that the receiving task still needs, so its utility depends on whether the retained content supports the judgments and actions required by the current task. While the quantitative ablations show that corrective experience and cleaned traces are more effective when used together, the mechanism underlying this complementarity is supported mainly by qualitative case analyses. At task start, VetEx injects both corrective experience and retrieved successful trajectories, resulting in more experience-related context than some reference baselines (Table 1; Appendix E). Although experience construction is relatively inexpensive, this additional context increases input-token consumption during task execution. Finally, our evaluation considers only single-round offline construction with fixed experience assets and does not examine continual experience updates.

5 Conclusion

We introduced VetEx, a training-free experience-reuse framework for frozen LLM agents that gates experience reuse according to the receiver's evidence gaps and the validity of task-reference trajectories. Across three benchmarks and three frozen receiver models, VetEx outperforms ReAct in all nine settings and achieves a higher three-run mean than the strongest reproduced experience-learning baseline in each setting. Our analyses show that more experience is not necessarily more useful: source expansion helps when it fills missing positive behavioral evidence, while more complex experience organization or stronger compiler models do not consistently improve downstream utility. We further find that cleaned task references are most effective when combined with corrective experience, suggesting that abstract behavioral constraints and concrete execution paths provide complementary forms of support.

AI USE STATEMENT

In this work, we used a generative AI tool (i.e., ChatGPT) to assist with polishing the manuscript, including checking spelling errors, reviewing the logic of the exposition, correcting grammatical errors, improving language clarity. We also used it for conducting literature search and related-work discovery. For research execution, generative AI tool was used to review code, provide figure-layout plans and design references. All AI-assisted revisions were carefully reviewed, verified, and revised by us. We take full responsibility for the final content of this work, including all text, claims, and related outputs.

REPRODUCIBILITY STATEMENT

Upon acceptance, we will release the VetEx code, including the experience-construction, coverage-gating, trajectory-cleaning, native-environment replay-validation, and task-start context-injection pipelines, together with the prompt templates, data structures, and evaluation scripts.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Zhicheng Cai, Xinyuan Guo, Yu Pei, Jiangtao Feng, Jinsong Su, Jiangjie Chen, Ya-Qin Zhang, Wei-Ying Ma, Mingxuan Wang, and Hao Zhou. Flex: Continuous agent evolution via forward learning from experience. *arXiv preprint arXiv:2511.06449*, 2025.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. Automanual: Constructing instruction manuals by llm agents via interactive environmental learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Tao Feng, Chongrui Ye, Tianyang Luo, Jingjun Xu, Xueqiang Xu, Haozhen Zhang, Zhigang Hua, Yan Xie, Shuang Yang, Ge Liu, and Jiaxuan You. Expgraph: Model-agnostic experience learning with graph-structured memory for llm agents. *arXiv preprint arXiv:2605.30712*, 2026.

Dayuan Fu, Biqing Qi, Yihuai Gao, Che Jiang, Guanting Dong, and Bowen Zhou. Msi-agent: Incorporating multi-scale insight into embodied agents for superior planning and decision-making. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 643–659, 2024a.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. *Advances in Neural Information Processing Systems*, 37:119919–119948, 2024b.

Jinglong Gao, Xiao Ding, Lingxiao Zou, Bibo Cai, Bing Qin, and Ting Liu. Expetrans: Llms are experiential transfer learners. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10577–10616, 2025.

Priyanshu Gupta, Shashank Kirtania, Ananya Singha, Sumit Gulwani, Arjun Radhakrishna, Gustavo Soares, and Sherry Shi. Metareflection: Learning instructions for language agents using past reflections. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8369–8385, 2024.

Hao-Lun Hsu, Nikki Lijing Kuang, Boyi Liu, Zhewei Yao, and Yuxiong He. Organize then retrieve: Hierarchical memory navigation for efficient agents. *arXiv preprint arXiv:2606.11680*, 2026.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 13358–13376, 2023.

Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. RAP: Retrieval-augmented planning with contextual memory for multimodal LLM agents. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

Qingcan Kang, Mingyang Liu, Shixiong Kai, Kaichao Liang, Zhentao Tang, Yuqi Cui, Tao Zhong, and Mingxuan Yuan. Retain or consolidate? budget-dependent operator selection for language agent memory. *arXiv preprint arXiv:2607.17545*, 2026.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024a.

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, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. In *The Twelfth International Conference on Learning Representations*, 2024b.

Yitao Liu, Chenglei Si, Karthik R. Narasimhan, and Shunyu Yao. Contextual experience replay for self-improvement of language agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14179–14198, 2025.

Yuchen Ma, Yue Huang, Han Bao, Haomin Zhuang, Swadheen Shukla, Michel Galley, Xiangliang Zhang, and Stefan Feuerriegel. Skillgen: Verified inference-time agent skill synthesis. *arXiv preprint arXiv:2605.10999*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2022.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *The Fourteenth International Conference on Learning Representations*, 2026.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Chen Qian, Yufan Dang, Jiahao Li, Wei Liu, Zihao Xie, Yifei Wang, Weize Chen, Cheng Yang, Xin Cong, Xiaoyin Che, Zhiyuan Liu, and Maosong Sun. Experiential co-learning of software-developing agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5628–5640, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Yuval Shalev, Zifeng Ding, and Mateja Jamnik. Training language agents to learn from experience. *arXiv preprint arXiv:2605.20477*, 2026.

Yibo Shi, Jungang Li, Linghao Zhang, Zihao Dongfang, Biao Wu, Sicheng Tao, Yibo Yan, Chenxi Qin, Weiting Liu, Zhixin Lin, Hanqian Li, Yu Huang, Song Dai, Yonghua Hei, Yue Ding, Xiang Li, Shikang Wang, Chengdong Xu, Jingqi Liu, Xueying Ma, Zhiwen Zheng, Xiaofei Zhang, Bincheng Wang, Nichen Yang, Jie Wu, Lihua Tian, Chen Li, and Xuming Hu. Androtmem: From interaction trajectories to anchored memory in long-horizon gui agents. *arXiv preprint arXiv:2603.18429*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan O Arik. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments. In *The Thirteenth International Conference on Learning Representations*, 2025.

Theodore Sumers, Shunyu Yao, Karthik R Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Forty-second International Conference on Machine Learning*, 2025.

Tianxin Wei, Ting-Wei Li, Zhining Liu, Xuying Ning, Ze Yang, Jiaru Zou, Zhichen Zeng, Ruizhong Qiu, Xiao Lin, Dongqi Fu, Zihao Li, Mengting Ai, Duo Zhou, Wenxuan Bao, Yunzhe Li, Gaotang Li, Cheng Qian, Yu Wang, Xiangru Tang, Yin Xiao, Liri Fang, Hui Liu, Xianfeng Tang, Yuji Zhang, Chi Wang, Jiaxuan You, Heng Ji, Hanghang Tong, and Jingrui He. A survey of agentic reasoning for large language models: Towards recursively self-improving and collective agents. *arXiv preprint arXiv:2601.12538*, 2026.

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, 2018.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Chongrui Ye, Yuxiang Liu, Yu Wang, Haofei Yu, Yining Zhao, Ge Liu, Julian McAuley, and Jiaxuan You. Auto-dreamer: Learning offline memory consolidation for language agents. *arXiv preprint arXiv:2605.20616*, 2026.

Jing Yu, Yibo Zhao, Jiaming Zhang, and Xiang Li. Lazymem: Retrieve broadly, construct selectively for efficient long-term agent memory. *arXiv preprint arXiv:2607.22690*, 2026.

Yunxiang Zhang, Yiheng Li, Ali Payani, and Lu Wang. Adamem: Test-time adaptive memory for language agents. In *Forty-third International Conference on Machine Learning*, 2026.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 19632–19642, 2024.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations*, 2024.

A Experimental Configuration

A.1 Benchmarks

ALFWorld. ALFWorld is an interactive household-manipulation benchmark that grounds language-model decisions in a text-based embodied environment (Shridhar et al., 2021). An agent must interpret a household instruction, inspect the current scene, and execute a sequence of navigation and object-manipulation actions until the requested state is reached. Success is determined by the environment's task-completion criterion.

HotpotQA. HotpotQA evaluates multi-hop question answering, where the agent must combine evidence from multiple documents to answer questions that cannot be resolved from a single fact (Yang et al., 2018). In our interactive adapter, the agent searches and reads evidence before emitting a final answer; an episode is correct only when that answer matches the reference under the official normalized exact-match criterion.

WebShop. WebShop is an interactive web-shopping benchmark in which an agent must search a large product catalogue, inspect candidate product pages, select required attributes, and complete the requested purchase action (Yao et al., 2022). The task therefore tests sequential web navigation, attribute grounding, and action selection rather than answer generation alone. We retain the native per-task reward and report its mean multiplied by 100.

Table 3: Benchmark configurations.

Benchmark	Tasks	Max step	Fold	Reported metric
ALFWorld	134	20	2	Success rate
HotpotQA	100	7	4	Official normalized EM
WebShop	100	15	4	Task-completion score

A.2 Baseline Implementations and Method-Specific Configurations

This subsection documents the implementations used for the baseline columns in Table 2.

Benchmark-specific adaptations. The baseline methods retain their original experience or memory logic; only the benchmark-facing interface is adapted where necessary. For ALFWorld, each method is rendered with the native task and observation format and its output is projected to one legal environment action under the native success and step-limit criteria. For HotpotQA, we wrap the methods in the finite-wiki interaction interface with the native *Search*, *Lookup*, and *Finish* actions, preserve the benchmark observation format, and apply the normalized exact-match criterion at termination. For WebShop, model responses are mapped to the native *search*, *click*, and *buy* actions using the controls visible on the current page, and episodes are scored with the native reward. When a method emits an additional structured wrapper or multiple candidate actions, the adapter extracts the first legal benchmark action before the environment step. These adapters change only parsing, rendering, and termination handling; they do not add method-specific experience or memory operations.

ReAct. ReAct interleaves language reasoning with environment actions and observations. We reproduce it with the native benchmark prompt and action interface, without offline experience construction, a memory bank, donor trajectories, or auxiliary model calls. At inference, the receiver sees only the native prompt and its current interaction history.

ExpeL. ExpeL extracts reusable insights from successful–failed experience and retrieves successful task demonstrations. We collect receiver trajectories on complementary folds, including the construction-time reflection and retry procedure, and then build frozen insight and successful-trajectory assets. The audited configuration has no donor channel. The reproduced full setting retains 20 rules and uses 2/6/2 retrieved examples for ALFWorld/HotpotQA/WebShop, respectively. Both the rules and retrieved successful trajectories are injected at task start.

ReasoningBank. ReasoningBank judges trajectories, extracts structured memory items, and retrieves memory for subsequent tasks. Our implementation follows the method's MaTTS construction, memory induction, and Top-1 retrieval procedures, including the official structured memory-item format and three fresh receiver rollouts per source task for parallel construction. The memory bank is built exclusively from complementary-fold evidence and is updated after each source task during construction. Before held-out evaluation, the bank is frozen; each held-out task receives one Top-1 retrieved memory entry and one temperature-zero rollout.

Synapse. Synapse treats successful trajectories as task-matched exemplars and retrieves relevant trajectories by similarity over task metadata. We reproduce its exemplar memory, trajectory-as-exemplar prompting, and similarity-retrieval pipeline following the original paper. The exemplar pool uses only successful receiver trajectories from complementary folds; no additional policy rollouts are performed when constructing the pool. For each held-out task, the system retrieves two task-matched successful trajectories from the FAISS index and injects the complete trajectories into the receiver's task context. Except for the embedding model, all construction, retrieval, and injection procedures follow the original paper. The original paper uses *text-embedding-ada-002*; because of the available API channel, our implementation uses BGE-M3 as the embedding model while retaining the FAISS memory and similarity-retrieval pipeline.

AdaMEM. AdaMEM indexes successful experience by environment state and synthesizes or refreshes a strategy during execution. We build the state-indexed bank from successful receiver trajectories on complementary folds, without donor trajectories in the audited construction. The reproduction uses the LOW controller, Top-1 state retrieval, and separate strategy and action calls. The external bank is frozen, while within-episode strategy refresh remains enabled.

SkillGen. SkillGen induces a reusable skill document from historical successful and failed behavior. We use historical complementary-fold assets for the headline comparison. The original SkillGen pipeline includes induction, generation, paired verification, iterative refinement, and net-gain selection; our full bridge implements these construction stages. The headline comparison uses a generated static skill artifact, injected as a frozen prefix at task start. Because of API access constraints, the implementation replaces the original `text-embedding-3-small` with Qwen3-Embedding-4B.

A.3 Experimental Configuration

Table 4: Experimental configuration.

Base configuration	
Receiver models	`DeepSeek-V3.2`, `Qwen3.6-27B`, `GLM-5.1`
Historical rollout temperatures	0.0, 0.3, and 0.8 for each benchmark
Evaluation decoding	Temperature $= 0$; greedy decoding; one rollout per task and repetition
DeepSeek receiver	Donor: `GLM-5.1`
Qwen receiver	Donor: `GLM-5.1`
GLM receiver	Donor: `Qwen3.6-27B`
Experience construction	
Constructor	`GLM-5.1`; temperature $= 0.2$
Reconciler	`GLM-5.1`; temperature $= 0.2$
Embedding model	`BAAI/bge-m3`
Retrieval and trajectory cleaning	
Query embedding model	`all-mpnet-base-v2`
Trajectory embedding model	`all-mpnet-base-v2`
Top-K	2
Cleaning model	`GLM-5.1`; temperature $= 0$

B VetEx Implementation Details

B.1 Pipeline Overview

Section 3.2 and Figure 3 describe the overall VetEx pipeline. Algorithm 1 summarizes the fold-safe offline construction, task-start retrieval, trajectory cleaning, replay validation, and initial-context assembly before the detailed protocols below. The high-level operators and symbols used by Algorithm 1 are summarized in Table 5.

Algorithm 1 VetEx: paired experience construction and replay-validated task-reference trajectory preparation.

Require: Fold-safe records $\mathcal{D}_r^+, \mathcal{D}_r^-$, optional donor successful trajectories $\mathcal{D}_p^+$, task q
Require: Frozen compiler C and benchmark adapter $\Pi_{\mathcal{E}}$
Ensure: All experience G_r^{all} and initial context P_q

Offline: paired corrective-experience construction

$\mathcal{P}_{\text{self}} \leftarrow \{(d^-, d^+) : d^- \in \mathcal{D}_r^-,\ d^+ \in \mathcal{D}_r^+,\ t(d^-) = t(d^+)\}$
$\mathcal{P}_{\text{peer}} \leftarrow \{(d^-, d^+) : d^- \in \mathcal{D}_r^-,\ d^+ \in \mathcal{D}_p^+,\ t(d^-) = t(d^+),\ t(d^-) \in \mathcal{U}_r\}$
$G_r^{\text{all}} \leftarrow \text{NamespacedUnion}(\text{CompileAndReconcile}(C, \mathcal{P}_{\text{self}}), \text{CompileAndReconcile}(C, \mathcal{P}_{\text{peer}}))$

Before task execution: reference-trajectory retrieval and cleaning

$R_q \leftarrow \text{RetrieveTask}(q, \mathcal{D}_r^+ \cup \mathcal{D}_p^+, K{=}2)$
for all $\tau \in R_q$ in retrieval order **do**
    $\tilde{\tau} \leftarrow \text{Clean}_{\mathcal{E}}(\text{Parse}_{\mathcal{E}}(\tau))$
    $\hat{\tau} \leftarrow \text{Replay}_{\mathcal{E}}(\tilde{\tau}, \tau)$ ▷ compact if validated; raw fallback otherwise
    $v_\tau \leftarrow \text{Render}(\hat{\tau})$
end for
$V_q \leftarrow (v_\tau)_{\tau \in R_q}$ ▷ In retrieval order

Initial-context construction

$P_q \leftarrow \text{Concat}(P_0, G_r^{\text{all}}, V_q)$
return G_r^{all} and P_q

Table 5: High-level operator contracts and notation used in Algorithm 1.

Function / symbol	**Interface contract or meaning**
FormPairs	Applies Self-first admission: pairs receiver failure records with same-task receiver success records, and uses donor success records only when the receiver has no same-task success record.
CompileAndReconcile	Applies the frozen compiler to one channel, removes duplicate formulations, resolves conflicts, and retains that channel's provenance links.
NamespacedUnion	Combines the Self and Peer outputs into All without erasing their source namespaces.
RetrieveTask	Ranks the eligible bank by task similarity and returns the first K candidates.
$\text{Parse}_{\mathcal{E}}$	Reads a candidate in the benchmark-native trace format.
$\text{Clean}_{\mathcal{E}}$	Removes unsupported or non-retainable steps without changing candidate identity.
$\text{Replay}_{\mathcal{E}}$	Replays the retained action chain in the native environment and returns a validated compact candidate or the stored raw candidate on rejection; the validation predicate is defined in Appendix B.3.3.
Render	Converts the validated compact or raw fallback trajectory into the benchmark-native task-reference format.
P_0	Benchmark-native base prompt.
G_r^{all}	Final All bank formed from the Self and Peer channels.
R_q	Ranked Top-K candidate set for task q.
V_q	Task-reference views arranged in retrieval order.
P_q	Initial context supplied to the frozen receiver.
$\Pi_{\mathcal{E}}$	Benchmark-adapter interface supplying parsing, cleaning, replay validation, and rendering.

B.2 Experience Construction and Representation

Section 3.2 and Figure 3 describe the overall experience-construction pipeline, while the experimental section explains its role in the controlled comparisons. This subsection records the implementation details needed to reproduce the experiments.

B.2.1 Constructor

The following cards reproduce the complete template used for each pair block. The constructor is frozen, with its model and temperature fixed as listed in Table 4.

Experience Compiler – System

```
You extract reusable, task-agnostic experience items from REAL agent trajectories
in {bench}/{vertical}, by comparing what high-scoring (success) runs did
vs low-scoring (failure) runs. Update the current experience set using 3 operators.

== Operators (choose per experience item) ==
- ADD <experience item>: a new reusable pattern. Prefer ADD whenever the behavior
  differs even slightly; cite >=1 trace_id from THIS block.
- UPVOTE <id>: existing experience item confirmed by this block; frequency only.
- DOWNVOTE <id>: existing experience item is factually wrong or ungrounded.
== HARD CONSTRAINTS ==
0. Write title and body in English.
1. Keep experience items task-agnostic; do not copy entity names, answers, files,
   exact locations, product IDs, or one-off routes.
2. Describe thinking/action-category behavior, never a specific code line.
3. Ground every rule in observations: a do rule must occur in the success
   trace; an avoid rule must describe behavior observed in the failure trace.
4. Do not write when_to_use; it is generated later with full context.
5. Preserve uncertainty with usually/tends to; avoid must/always.
6. When in doubt, ADD rather than skip; deduplication happens later.
```

Experience Compiler – Input

```
== Current experience set ==
{current_experience_items}

== Trajectory block ({mode}) ==
{block}
```

JSON / Schema

```
== OUTPUT (JSON object only) ==
{"ops":[
  {"op":"ADD","experience":{"polarity":"do|avoid","title":"...",
    "body":"..."},"cited_signal_ids":["<trace_id>"]},
  {"op":"UPVOTE","target_id":"..."},
  {"op":"DOWNVOTE","target_id":"..."}
]}
```

The pair input is rendered as ‘TASK‘, followed by labelled ‘SUCCESS‘ and ‘FAILURE‘ traces. Each step contains its thinking subtype (when present), action code (capped at 160 characters), thought (capped at 200 characters), and observation (capped at 200 characters). The trace is otherwise complete and retains step order. The parser rejects malformed JSON, unknown trace IDs, task-specific text, and operations that violate the cited-pair constraint; rejected calls are retried without changing the evidence block.

B.2.2 Reconciler and Deduplication

After pairwise construction, each channel is reconciled independently before the Self and Peer namespaces are united. The operation is not a single global summarization call. It is a three-stage, recall-first aggregation pipeline: inexpensive semantic candidate generation, an LLM screening pass, and a trace-grounded adjudication pass. The intermediate sets below are channel-local; they are not

additional experience banks and are discarded after the final channel output is rendered. The fragments below show the actual data flow and omit transport, retry, and logging boilerplate.

Stage 1: embedding candidate generation. The input is one channel's compiled item set T_c (either Self or Peer), where each item contains its title, body, polarity, and cited source-signal IDs. The output is a set of candidate connected components C_1; singleton items are omitted because they cannot require reconciliation. The implementation concatenates each title with the first 400 characters of its body, embeds the texts with *BAAI/bge-m3*, computes cosine similarity, and connects pairs whose similarity is at least 0.65. A union–find pass returns the connected components. This stage is deliberately over-inclusive: semantic similarity only proposes items for inspection and never deletes or merges an item.

For this stage, the channel-local records are projected to the JSON fields below; the implementation uses the existing *experience_id* as the item key. The remaining provenance fields in the canonical experience-item record are not read by candidate generation.

JSON / Schema

```
STAGE 1 INPUT (JSON object; no LLM call)
{
  "items": [
    {
      "experience_id": "<string>",
      "polarity": "do|avoid",
      "title": "<string>",
      "body": "<string>",
      "source_signal_ids": ["<trace_id>", "..."]
    }
  ]
}

STAGE 1 OUTPUT (JSON object)
{
  "components": [
    ["<experience_id>", "<experience_id>", "..."]
  ]
}
```

Stage 2: recall-first screening. Stage 2 produces a disjoint set of suspected components C_2, each containing the item IDs that should receive trace-grounded adjudication. It keeps a group when its members may be duplicates, offer opposing guidance, or concern the same decision point, and it is instructed to keep uncertain cases and optionally split a broad component into smaller subgroups. The parser then removes unknown IDs, groups with fewer than two items, and overlapping groups. Thus, this stage controls the expensive trace-grounded calls without treating the screener's judgment as a final merge decision.

The following is the actual system prompt used for this call (the inserted *groups* block is shown separately below). It is intentionally recall-first: a possible duplicate or conflict is passed to Stage 3 rather than discarded at this point.

Reconciler – System

```
You are a RECALL-FIRST screener for experience-item merge/conflict clusters.
Below are candidate groups pre-selected by embedding similarity (topically near). A later
     stage
pulls the original traces and makes the precise call -- your ONLY job is to NOT lose
     anything worth
checking. KEEP a group (or a >=2 sub-group) if its members have ANY of: look like
     duplicates /
restatements; give opposing guidance (broaden vs don't-broaden, commit-now vs keep-
     investigating);
act on the same decision point. Only DROP a pair when confident they are truly independent.
When in doubt, KEEP. You may split a candidate group into smaller suspected sub-groups.

== Candidate groups ==
{groups}
```

```
== Output (JSON only) ==
{{"merges":[{{"ids":["id1","id2",...]}}, ...]}}
Only ids. Bias toward keeping; the next stage filters false positives cheaply.
```

JSON / Schema

```
STAGE 2 OUTPUT (JSON object only)
{"merges":[{"ids":["id1","id2",...]}]}
```

Stage 3: trace-grounded adjudication and deterministic application. The input is one suspected component $g \in C_2$, the original item records, and the source traces referenced by their cited signal IDs. For each member, the implementation retrieves up to two cited traces and renders the original trajectory evidence together with the abstracted item. The GLM-5.1 adjudicator returns one of four verdicts: *duplicate*, *tension*, *pseudo_conflict*, or *unrelated*.

The verdict is then applied by code rather than by trusting free-form model output. A *duplicate* component is collapsed to one reconciler-provided item, with maximum priority and importance and the union of source-signal IDs. A *tension* is replaced by one conditional item with explicit branches and the same provenance union. For a *pseudo_conflict*, any explicitly identified duplicate subgroups are collapsed first; the remaining items stay separate, with their applicability conditions restored. An *unrelated* component is left unchanged. Invalid JSON is retried; if adjudication still fails, the component is retained unchanged. No verdict can add a new trace, task fact, or provenance link.

The adjudicator uses the following system prompt. Unlike Stage 2, this call receives the source trajectory excerpts that induced each item, so that a superficially conflicting pair can be separated when the traces show different sub-operations.

Reconciler – System

```
You verify whether these agent experience items are DUPLICATE / TENSION / PSEUDO_CONFLICT /
UNRELATED -- by examining the ORIGINAL traces each was abstracted from. The experience
    items' own text had
task-specific context stripped during abstraction; the traces restore it. Use the traces to
     find
WHICH decision point or sub-operation each experience item actually addresses.

== Members (experience-item text + the trace excerpts it was induced from) ==
{members}

== Definitions ==
- duplicate: same behavioral rule, just reworded.
- tension: they give OPPOSING guidance at the SAME decision point (a genuine conflict to
    reconcile).
- pseudo_conflict: they look opposing but the traces show they act on DIFFERENT sub-
    operations /
  contexts (each correct in its own) -- the apparent conflict is only because context was
    stripped.
- unrelated: not actually the same sub-problem.

== Output (JSON only) ==
{{"verdict":"duplicate|tension|pseudo_conflict|unrelated",
  "reasoning":"<2-3 sentences citing what the traces show>",
  "merged":{{"polarity":"do","title":"...","when_to_use":"...","body":"..."}},
  "distinguishing_context":{{"<experience_id>":"<restored when_to_use for this one>"}},
  "duplicate_subgroups":[{{"ids":["<id>","<id>"],"merged":{{"polarity":"do","title":"...","
    when_to_use":"...","body":"..."}}}}],
  "synthesis":{{"polarity":"do","title":"...","when_to_use":"...","body":"IF <A> THEN <a>;
    IF <B> THEN <b>. ..."}}}}
(merged required ONLY for duplicate; distinguishing_context required ONLY for
    pseudo_conflict;
duplicate_subgroups are optional for pseudo_conflict; synthesis is required ONLY for
    tension.)
EVERY when_to_use you output (merged / distinguishing_context / subgroup merged / synthesis
    ) MUST
follow: {style}
```

Experience Compiler – Input

```
### <experience_id> [do|avoid] <title>
body: <abstracted experience body>
-- induced from traces --
[<trace_id>] score=<source score>
<formatted source trajectory, preserving step order>

The renderer includes at most two cited source traces per member.
```

JSON / Schema

```
STAGE 3 OUTPUT (JSON object only)
{"verdict":"duplicate|tension|pseudo_conflict|unrelated",
 "reasoning":"...",
 "merged":{...},
 "distinguishing_context":{...},
 "duplicate_subgroups":[...],
 "synthesis":{...}}
```

After all suspected components in a channel have been processed, the surviving channel-local items are rendered with their updated provenance. The reconciled Self and Peer outputs are then combined by the namespaced union in Algorithm 1 to form G_r^{all}; no cross-channel merge is performed at this point.

For both LLM stages, malformed or schema-invalid responses are retried with the same evidence block and an output-only reminder. The final parser removes unknown IDs and overlapping groups. The duplicate, tension, pseudo-conflict, and unrelated actions above are deterministic code paths; the LLM never adds a source trace or provenance link that was absent from the input.

B.2.3 Experience Representation

An experience item stores its applicability condition, title, body, polarity, priority, and provenance. The machine-readable example below shows selected fields of item *I_003* from the Qwen–HotpotQA fold-1 Peer bank; fields not used in the injected text remain available for audit.

JSON / Schema

```
{
  "experience_id": "peer::I_003",
  "polarity": "do",
  "priority": 19,
  "when_to_use": "When a search query fails and returns a list of similar entities.",
  "title": "Systematically process similar-result candidates by prioritizing, searching
     directly, and verifying constraints",
  "body": "Prioritize candidates whose category matches the question's entity type. Search
     each promising candidate directly (not via a combined query) to verify all its
     attributes against the question constraints, rather than guessing based on prior
     knowledge. Exhaust the list of similar candidates before pivoting to a new search
     query.",
  "source_role": "peer",
  "receiver_model": "qwen3.6-27b"
}
```

source_role is *self* or *peer*, and *pair_id* preserves the ordered success–failure pair. The canonical schema additionally stores *source_signal_ids*, *supporting_pair_ids*, and evidence statistics. In all three benchmark adapters, the frozen evaluation assets are injected as a common benchmark-specific header followed by one complete line per experience item:

Injected Template

```
# Tips learned from past {benchmark} attempts (apply when relevant):
- [{DO|AVOID}] {when_to_use} --- {title} --- {body}
```

The polarity determines the *DO* or *AVOID* tag. The applicability condition, title, and complete body are visible to the receiver; identifiers, source links, provenance, and other audit fields are not injected.

Rendered experience-item instance. The following line renders the same item in the format above. Its condition, title, and body are reproduced verbatim from the saved item.

Injected Template

```
- [DO] When a search query fails and returns a list of similar entities. ---
Systematically process similar-result candidates by prioritizing, searching
directly, and verifying constraints --- Prioritize candidates whose category
matches the question's entity type. Search each promising candidate directly
(not via a combined query) to verify all its attributes against the question
constraints, rather than guessing based on prior knowledge. Exhaust the list
of similar candidates before pivoting to a new search query.
```

B.3 Task-Start Trajectory Retrieval, Preparation, and Injection

Before the receiver's first action, all experience items constructed from the complementary folds form the All experience prefix. The following three subsections specify the task-start trajectory interface in the order in which it is executed; the complete flow is summarized in Section 3.3 and Figure 3.

B.3.1 Task-matched retrieval

Task-reference retrieval uses the local 'all-mpnet-base-v2' Sentence-Transformers encoder with normalized vectors and exact cosine via inner product. For an embedding $h(x)$, we use the L_2-normalized vector $\bar{h}(x) = h(x)/\|h(x)\|_2$ and score a query–candidate pair by

$$\mathrm{sim}(x, y) = \frac{h(x)^\top h(y)}{\|h(x)\|_2 \, \|h(y)\|_2} = \bar{h}(x)^\top \bar{h}(y).$$

Thus, *exact cosine* means an exact inner-product computation on the normalized vectors; no approximate-nearest-neighbor index or similarity threshold is used for retrieval.

The query is the benchmark adapter's task-similarity text, and candidates are drawn from the successful-trajectory bank of complementary folds. The index contains only complete trajectories satisfying the native success criterion; for WebShop, this means reward $= 1.0$. Candidates are sorted by descending score with a stable index tie-break, and the first $K = 2$ successful traces are retained. Only these selected Top-2 candidates are passed to the preparation stage. Retrieval is not LLM-reranked and is not repeated within an episode. Candidate identity, source fold, and retrieval position remain unchanged throughout preparation and injection.

B.3.2 Trajectory cleaning

The cleaning step is performed by a fixed LLM selector. It receives one retrieved successful trajectory and outputs the source step IDs of the external actions to retain. The selected actions must come from the successful trace, preserve their original order, and must not be generated, rewritten, or reordered. In the reported experiments, the selector uses the same frozen model as experience extraction at temperature 0.

The selector preserves information needed by later actions based on the task description and earlier observations. An action must be retained if its observation first reveals a later search term, entity, location, product identifier, option, or other action argument. Cleaning removes only exploratory detours, repeated searches, and other redundant steps that are unnecessary for later actions or their informational prerequisites, while retaining the source trajectory's final external action. After source step IDs and action formats are checked, the corresponding content is organized in source-trajectory order. The corresponding trajectory-cleaning prompt is as follows:

Trace Cleaner – Selector

```
You select a compact, executable view of one REAL successful agent trajectory.

SELECTION CONTEXT success_only

TASK {task_text}

SUCCESS TRACE (select from this trace only) {positive_trace}

RULES 1. Output only step ids that already exist in SUCCESS TRACE and contain a real
     external
action. 2. If a step id has multiple occurrences, select it only when exactly one
     occurrence is
marked external_action: true. 3. Keep the selected ids in their original, strictly
     increasing
order. 4. Preserve epistemic prerequisites. Never delete an action whose Observation first
reveals a later retained search term, entity, location, ASIN, option, answer candidate, or
     other
action argument. 5. Every argument in a retained action must be available from TASK or an
earlier retained Observation. Do not use hindsight from later observations or from knowing
     that
the source trajectory succeeded. 6. Select every action needed for the successful route to
remain executable from the task's initial state. 7. Remove only exploratory detours,
     repeated searches, and
other actions that are not needed for execution or for establishing the knowledge required
     by a
later retained action. 8. Find the LAST step in SUCCESS TRACE marked external_action: true.
      You
MUST include that exact step_id as the final selected id, even when its action is Finish,
submit, purchase, or another seemingly obvious terminal action. 9. Do not invent, rewrite,
normalize, or paraphrase an action. 10. The rationale is synthetic and is used only for
     audit.
Briefly name the major stages (for example, discover -> verify -> finish); do not repeat
     long
trajectory thoughts.

OUTPUT JSON ONLY
{"selected_source_step_ids":[1,2,3],"synthetic_rationale":"Discover the needed information,
      verify it, then execute the original successful terminal action."}
```

B.3.3 NATIVE REPLAY VALIDATION

Given a retrieved source trajectory τ and the step IDs selected by the cleaner, the benchmark adapter replays the corresponding original actions in the environment associated with the source task. The actor is disabled during this check. Each action is reconstructed from the stored source action and submitted using the benchmark's native action format; no new action is generated by the receiver during replay.

We mark the compact candidate as *validated* only when every selected action is accepted and executed by the adapter and the replay reaches the benchmark's official terminal success criterion for the source task. Action parseability or individual action legality alone is insufficient for validation. The observations returned during replay are used to advance and validate the source-environment state. They are not treated as observations from the current target task and do not replace the receiver's online observations.

If any selected action cannot be parsed, is rejected by the environment, fails to execute, or the replay does not satisfy the terminal success criterion, the compact candidate is rejected and the stored raw source trace is retained. Infrastructure errors, exceptions, and replay-budget exhaustion are handled through the same raw fallback.

B.3.4 TASK-START INJECTION

Before the receiver's first action, the system renders the final trajectory views corresponding to the Top-2 candidates as task references. Candidate count, identity, source, and slot order remain unchanged. Task-reference trajectories and All corrective experience jointly form the receiver's initial context, providing concrete task paths and cross-task behavioral guidance, respectively.

The task-reference block uses the benchmark adapter's example format:

Injected Template

```
TASK-SIMILARITY SUCCESSFUL TRAJECTORIES
The following are complete solutions to similar tasks. Reuse their search
order and action pattern when relevant, while grounding exact entities in
the current observations.

RETRIEVED SUCCESSFUL EXAMPLE 1
Task: {task_embedding_text}
Thought: {thought}
Action: {action}
Observation: {observation}
```

ALFWorld uses its native *Your task is to:* / *think:* rendering, while HotpotQA and WebShop use the labelled form above. Retrieved task traces are not re-sliced at injection time; their stored successful steps are rendered in order, subject only to the benchmark's native prompt limits. The two selected references retain their retrieval order.

C Experience Construction Ablations

C.1 Repeated-Evaluation Robustness

The table below reports the current R=3 repeated-evaluation matrix. Each R=3 cell is the mean and sample standard deviation over three complete, protocol-matched temperature-zero runs. We include the ReAct baseline, all core VetEx configurations, and the strongest external baseline selected by its single-run score in Table 2 for each benchmark–receiver-model cell. The table summarizes repeatability; the task-level audit of the core trace contrast is reported in Appendix D.1.

Table 6: Current R=3 robustness summary (%, mean ± sample SD). "Ext." is the selected external baseline from Table 2.

Benchmark	Control	External baselines			VetEx variants & full method		
	ReAct	Ext.	Raw-only	Clean-only	VetEx-E	VetEx-Exp+RT	**VetEx**
DeepSeek-V3.2							
ALFWorld	54.47 ± 3.87	82.09 ± 5.63	82.84 ± 1.97	80.85 ± 0.86	75.87 ± 1.17	85.57 ± 1.17	**90.55 ± 0.43**
HotpotQA	35.33 ± 3.79	46.67 ± 1.53	41.00 ± 3.46	40.33 ± 3.06	47.67 ± 1.15	46.67 ± 3.51	**48.67 ± 1.53**
WebShop	42.94 ± 1.94	41.69 ± 5.95	40.76 ± 1.45	41.89 ± 1.69	53.66 ± 1.25	53.38 ± 0.81	**60.47 ± 1.93**
Qwen3.6-27B							
ALFWorld	82.33 ± 1.55	90.55 ± 1.55	87.06 ± 2.28	82.59 ± 1.14	91.30 ± 0.87	91.04 ± 0.00	**92.04 ± 0.43**
HotpotQA	37.00 ± 2.00	45.00 ± 1.00	41.33 ± 1.53	43.00 ± 0.00	45.00 ± 3.61	47.00 ± 1.00	**47.33 ± 0.58**
WebShop	53.76 ± 3.34	58.84 ± 1.37	54.71 ± 2.57	53.95 ± 0.99	59.54 ± 0.59	60.69 ± 0.93	**64.72 ± 1.63**
GLM-5.1							
ALFWorld	84.60 ± 0.87	91.53 ± 2.14	90.80 ± 2.15	89.80 ± 0.86	90.80 ± 0.43	94.27 ± 1.57	**93.78 ± 1.87**
HotpotQA	46.00 ± 1.73	51.33 ± 3.21	48.33 ± 2.08	48.67 ± 2.89	52.33 ± 1.15	49.00 ± 2.65	**51.67 ± 2.08**
WebShop	48.66 ± 0.56	56.55 ± 2.91	54.09 ± 0.20	57.02 ± 1.40	60.45 ± 1.50	58.07 ± 1.27	**61.07 ± 0.79**

C.2 Paired Source-Composition Audits

Table 7 is a complete 3 × 3 audit of source expansion. Each row pairs Self (receiver-only sources) with All (adding Peer Experience from donor success records for tasks not covered by Self) by exact task ID within the same fold, so the comparison measures the change caused by source expansion rather than a ReAct construction contrast. ALFWorld and HotpotQA report successful/EM task counts and list repairs, regressions, and ties. Because WebShop has continuous rewards, it reports mean reward multiplied by 100 and lists per-task increases, decreases, and ties instead.

Table 7: Complete 3 × 3 paired source-expansion audit. Repair, regression, and tie counts denote tasks solved only by All, only by Self, or by both/neither, respectively, for ALFWorld and HotpotQA; for WebShop they denote per-task reward increases, decreases, and ties. HotpotQA uses the matched 7-step protocol.

Benchmark	Source banks			Paired outcomes		
	Self	All	Δ (pp)	Repair	Regression	Ties
DeepSeek-V3.2						
ALFWorld	97/134	100/134	+2.2	15	12	107
HotpotQA	46/100	47/100	+1.0	4	3	93
WebShop	49.29	55.08	+5.79	17	7	76
Qwen3.6-27B						
ALFWorld	118/134	121/134	+2.2	6	3	125
HotpotQA	45/100	41/100	−4.0	4	8	88
WebShop	54.40	59.40	+5.00	10	4	86
GLM-5.1						
ALFWorld	122/134	122/134	+0.0	2	2	130
HotpotQA	46/100	51/100	+5.0	6	1	93
WebShop	55.49	61.37	+5.88	15	11	74

C.3 Experience-Novelty Audit for Source Expansion

Table 8 summarizes source expansion across all nine settings. Across the nine settings, the manually labeled donor-added items comprise 64 restatements, 173 refinements, and 156 new corrective themes. We compare each donor-derived experience item with the available Self Experience in the same held-out fold and assign one of three manual labels: *restatement* preserves the same failure trigger and corrective action; *refinement* narrows or operationalizes an existing correction without introducing a different behavior; and *new corrective theme* introduces a distinct trigger or corrective action. We read the title, applicability condition, and full corrective body.

Table 8: Complete nine-setting source-expansion summary. Donor composition reports manually coded restatements, refinements, and new corrective themes among donor-added experience items. Pair counts are fold-local same-task success–failure comparisons.

Benchmark	Pair counts		Experience items		Donor composition			Outcome
	Self	All	Self	All	Restatement	Refinement	New corrective themes	All−Self
DeepSeek-V3.2								
ALFWorld	372	576	71	114	9	20	14	+2.2
HotpotQA	144	306	89	141	2	15	35	+1.0
WebShop	144	234	68	131	12	21	30	+5.8
Qwen3.6-27B								
ALFWorld	156	225	14	30	7	3	6	+2.2
HotpotQA	90	315	56	162	11	71	24	−4.0
WebShop	81	117	31	57	4	8	14	+5.0
GLM-5.1								
ALFWorld	30	84	17	30	5	2	6	+0.0
HotpotQA	108	216	56	99	12	24	7	+5.0
WebShop	54	171	22	53	2	9	20	+5.9

Examples of the manual labels. The three cards below give one labeled example for each category from the GLM–ALFWorld fold-1 bank. Each card presents the titles and bodies of the Peer and Self experience items, followed by the manual rationale.

Type = Restatement

Peer experience	[AVOID] Do not repeat the same or similar actions after receiving a 'Nothing happens' response — When an action fails with a no-effect response, the agent should change its strategy rather than repeating the same or similar actions. The failure trace tried to take a non-existent item from a location, got a failed response, and then repeatedly issued navigation commands to the same location, getting completely stuck in a loop instead of exploring other locations.
Self experience	[DO] Abandon unresponsive locations and explore new ones — After a location returns negative or empty responses (e.g., 'Nothing happens'), stop retrying actions or stacking examination verbs at that location, and do not revisit it later in the trajectory. Instead, pivot immediately to a completely different, unvisited location to continue the search.
Reason	Same no-effect trigger and immediate strategy change.

	Type = Refinement
Peer experience	[DO] Prioritize semantically associated receptacles for target objects — Prioritize receptacles that are commonly associated with or likely to contain the target object type (e.g., cabinets for small objects, garbage cans for food items, drawers for personal items, toilets for hygiene items) as early search locations to yield faster finds.
Self experience	[AVOID] Conserve step budget by skipping low-yield locations during object search — When searching for an object, avoid exhaustively visiting every possible receptacle in sequence if some are unlikely to contain the target. The success trace went directly to the correct location, completing the task within the step limit. The failure trace wasted steps checking multiple low-probability locations—none of which contained the target—before finally reaching the correct one. Although it found and picked up the object, it ran out of steps before performing the final placement action. Exhaustive exploration of low-probability locations can consume the step budget needed for task-critical actions at the end.
Reason	Supplies a concrete prior for the existing high-yield-first search rule.

	Type = New corrective theme
Peer experience	[AVOID] Do not lose track of the final placement step after completing intermediate subgoals — After completing all intermediate subgoals, the agent must still execute the final placement. The failure trace successfully completed the intermediate steps but then failed to place the item in the target receptacle—getting sidetracked by re-examining receptacles instead of performing the placement action, leaving the task unfinished.
Self experience	[DO] Place objects in the specific receptacle instance where they were found (or a confirmed-accessible one) — When the task requires placing an object into a receptacle type, choose the specific instance that is known to be accessible—ideally the one the object was originally taken from. In the success trace, the agent placed the object in the receptacle where it had successfully picked it up, and the task was solved. In the failure trace, the agent chose a different instance for placement, which had already returned a negative response on an earlier open attempt, and the final open action again returned a negative response, causing failure. Receptacles previously confirmed inaccessible or empty should not be selected as placement targets.
Reason	Introduces a distinct terminal-step trigger and corrective action: execute the final placement after intermediate subgoals, rather than selecting a confirmed-accessible placement receptacle.

C.4 Case Study Of Source Expension

This section compares the source-expanded All condition with the receiver-only Self condition and provides complete task-level behavioral evidence supporting the source-expansion conclusion in Section 4.3. We examine three types of source-expansion cases. The positive case supplies a missing behavior and changes the target receiver accordingly; the null case adds construction material without adding a new behavioral direction; and the negative case fails to fill the relevant behavioral gap, instead inducing unnecessary follow-up search or delayed termination that causes a regression on the target task.

C.4.1 Positive-Evidence Expansion: Supplying Missing Behavior

We begin with the downstream failure that motivates the source-expansion case. The held-out task *fixed_63* asks Qwen to purchase capri pants with the *kahuna* color, *relaxed fit*, and size *20 plus*, at a price below 60 dollars. Under Self, Qwen checks several candidates but never reaches a product page that exposes the required color and size together, and never emits *Buy Now*. The missing behavior is to avoid rewriting the query from an unverified brand assumption when a plausible candidate can be inspected for its options.

The All bank contains a Peer Experience item aligned with this failure mode. Its source pair links Qwen receiver failure *WebShop__fixed_17__qwen3.6-27b__t0.8* to GLM donor success record *WebShop__fixed_17__glm-5.1__t0.8*. The decisive same-task contrast is shown below.

Source pair | WebShop | `fixed_17`

Question

The task requests an `mn4` foundation for sensitive skin, below 40 dollars.

Qwen receiver failure

Round 1

Action. `search[mn4 color foundation sensitive skin].`
Observation. The results include B077PR9TL4, but its title does not expose mn4.

Round 2

Thought. Treat mn4 as a likely L'Oreal shade and refine the query around that brand.

Round 3

Action. Search L'Oreal/MN4 variants, paginate, and issue another query from the results page.
Observation. The run produces an invalid/repeated action and exhausts its budget without opening B077PR9TL4.
Result. × Reward 0.

GLM donor success record

Round 1

Action. `search[mn4 color foundation sensitive skin].`
Observation. The results include B077PR9TL4.

Round 2

Thought. The shade is not in the snippet; inspect a promising candidate instead of adding a guessed brand.

Round 3

Action. `click[B077PR9TL4].`
Observation. The detail page exposes mn4 and size options.

Round 4

Action. `click[mn4]; click[1 fl oz (pack of 1)]; click[Buy Now].`
Observation. Score 1.0.
Result. Reward 1.

The Qwen failure never checks the candidate already exposed by the page; it binds the attribute code to a guessed brand and searches around that assumption. The GLM success retains the attribute term, opens the candidate, and confirms the option from the observation. The compiler extracts this local difference as *peer::WS_SELF_F1_002*. Its title and body are shown below:

Experience Compiler – Output

```
[AVOID] Avoid adding guessed brand names or assumptions to the search query --- Avoid
    searching with guessed brand names or assuming which brand a specific attribute code
    belongs to, as this tends to result in invalid or failed actions. Keep search queries
    concise and attribute-focused rather than guessing specific product titles or brands.
```

We first examine whether the newly added Peer Experience is already contained in the receiver model's Self Experience bank. Its source records are *WebShop__fixed_17__glm-5.1__t0.8* and *WebShop__fixed_17__qwen3.6-27b__t0.8*. In the fold-1 Self Experience bank, we find that, although it already contains adjacent rules about including exact attributes and inspecting candidate options, no rule requires avoiding inferring a brand from an attribute code and inserting that guess into the query. Therefore, this Peer Experience supplements a concrete behavioral direction not yet covered by the Self Experience bank, rather than simply repeating an existing rule. Table 9 lists all three Peer Experience items in this fold, their nearest Self items, and their relations to the corresponding Peer Experience items to substantiate this conclusion.

Table 9: Fold-1 Self–Peer comparison for Qwen–WebShop. Entries summarize the meaning of Self and Peer experience items; the full experience text is not shown. All three Peer items are paired with their nearest Self item. R and N denote restatement and new corrective theme.

Self Experience summary	Peer Experience summary	Relation
Include exact option values in the initial query	Include exact distinguishing attributes and identifiers	R
Include exact option values in the initial query	Avoid adding guessed brands or assumptions	N
Click promising results to explore options	Click promising products rather than browsing or refining	R

We then return to the held-out *fixed_63* failure to test whether this newly added behavior changes the receiver's actual decisions. The two columns below transcribe in full the action order and explicit-thought order under All and Self; long product-list observations are abbreviated, but no action or explicit thought is omitted.

Held-out target record | Qwen3.6-27B | `fixed_63`

Question

The task requests a `kahuna` capri pant with relaxed fit, size `20 plus`, and price below 60 dollars.

All

Round 1
Action. search[kahuna colored capri pant relaxed fit size 20 plus].
Observation. Results include B09PPMHNMS and B08SQHTMPR.

Round 2
Thought. Inspect the candidate matching relaxed fit and plus size.

Round 3
Action. click[B09PPMHNMS].
Observation. The page has 20 plus, but not kahuna.

Round 4
Thought. Return and inspect another related product.

Round 5
Action. click[Back to Search].

Round 6
Action. search[kahuna capri pant relaxed fit size 20 plus].
Observation. The results again include B08SQHTMPR.

Self

Round 1
Action. search[kahuna colored capri pant relaxed fit size 20 plus].
Observation. Results include several capri pants.

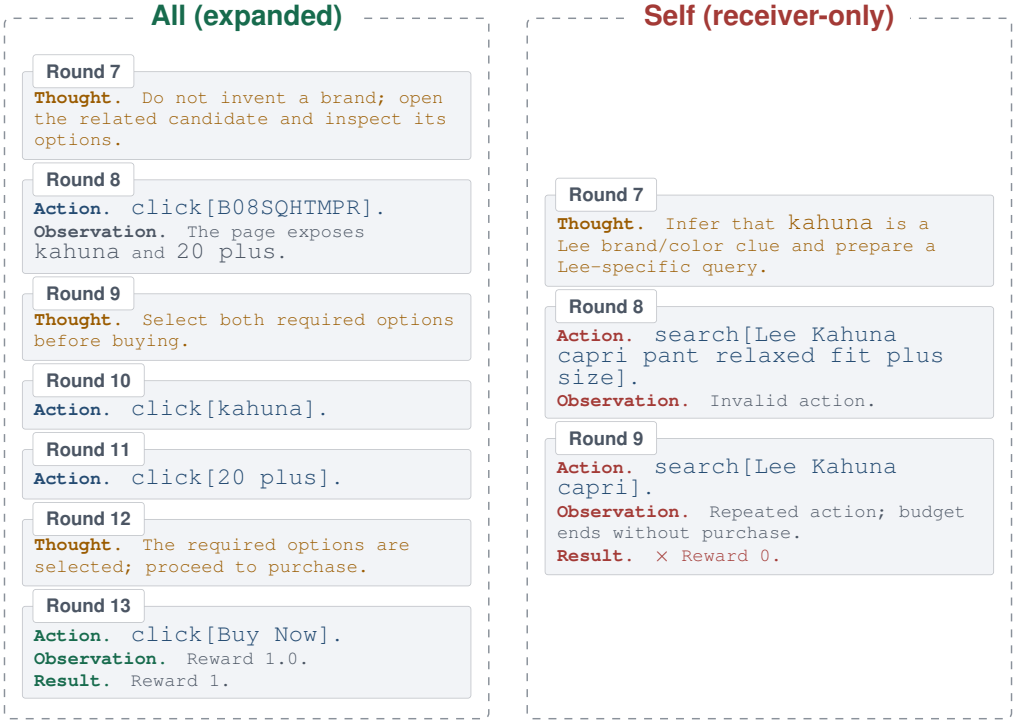
Round 2
Thought. Inspect a cheaper product whose title mentions relaxed and plus size.

Round 3
Action. click[B097RGT165].
Observation. No kahuna option is present.

Round 4
Thought. Return and search for a product with the missing color.

Round 5
Action. click[Back to Search].

Round 6
Action. search[kahuna capri pant relaxed fit plus size].
Observation. The results again include B08SQHTMPR.



On the held-out task, All does not follow Self's guessed-brand query path. It opens the related candidate already shown in the results, confirms *kahuna* and *20 plus* from the page, and completes the purchase. The downstream behavior changes in the direction specified by the added Peer Experience item.

C.4.2 No Transfer: Added Evidence Misses the Target Error

This section provides detailed behavioral evidence for the source-expansion relation examined in Section 4.3. The held-out fold-1 *Cloth–Cabinet* task exposes the remaining error. Both lanes ultimately fail after spending actions on the wrong object type, but their intermediate behavior differs. All initially treats the handtowel as a cloth (Round 3), attempts to clean it, and then recognizes the type error. It later finds and takes the actual cloth, but the run ends before it can clean and place it. Self first incorrectly believes that the cloth is on the empty holder (Round 2) and fails to take it. It then acquires the handtowel and unsuccessfully tries to clean it. Self sees the actual cloth in cabinet 4 in its final round, but the run ends before it can pick up, clean, or place the cloth.

Table 10: Behavioral coverage of the seven Peer Experience items in the GLM–ALFWorld fold-1 bank relative to the held-out *Cloth–Cabinet* error.

Behavior covered	Relation to the remaining error
Complete final placement	Not reached: the cloth is not cleaned before the budget ends.
Leave a failed-action loop	The run changes location after failures but still selects the wrong object type.
Use recognized commands	All recorded actions use valid command forms.
Search semantically likely receptacles	Changes location order, not the cloth–handtowel distinction.
Treat unlisted contents as empty	All leaves the empty holder, then misclassifies a visibly present handtowel.
Interleave planning and actions	Controls planning cadence, not object identity.
Avoid reopening accessible receptacles	Controls receptacle state, not object identity.

Held-out null result | GLM-5.1 | Cloth–Cabinet

Question

Clean a cloth and put it in a cabinet. Both conditions fail after spending the action budget on a handtowel that does not satisfy the requested object type.

All (expanded)

Round 1
Action. go to handtowelholder 1.
Observation. The holder is empty.

Round 2
Action. go to handtowelholder 2.
Observation. Handtowel 1 is visible.

Round 3
Thought. A handtowel is a type of cloth.

Round 4
Action. take handtowel 1; go to sinkbasins 1 and 2; try to clean it.
Observation. Both cleaning attempts return "Nothing happens".

Round 5
Thought. The task requires an actual cloth; search the cabinets.

Round 6
Action. Search cabinets 1-4.
Observation. Cloth 1 is visible in cabinet 4.

Round 7
Action. take cloth 1 from cabinet 4.
Observation. Nothing happens because handtowel 1 is still held.

Round 8
Action. put handtowel 1 in/on cabinet 4; take cloth 1 from cabinet 4.
Observation. The cloth is acquired, but the run ends before cleaning or placement.
Result. × Reward 0.

Self (receiver-only)

Round 1
Action. Check countertop 1 and handtowelholder 1.
Observation. No cloth is visible.

Round 2
Thought. Cloth 1 is on the empty holder.
Action. take cloth 1 from handtowelholder 1.
Observation. Nothing happens.

Round 3
Action. go to handtowelholder 2; take handtowel 1.
Observation. Handtowel 1 is acquired.

Round 4
Action. Go to sinkbasins 1 and 2 and try to clean the handtowel.
Observation. Both attempts return "Nothing happens".

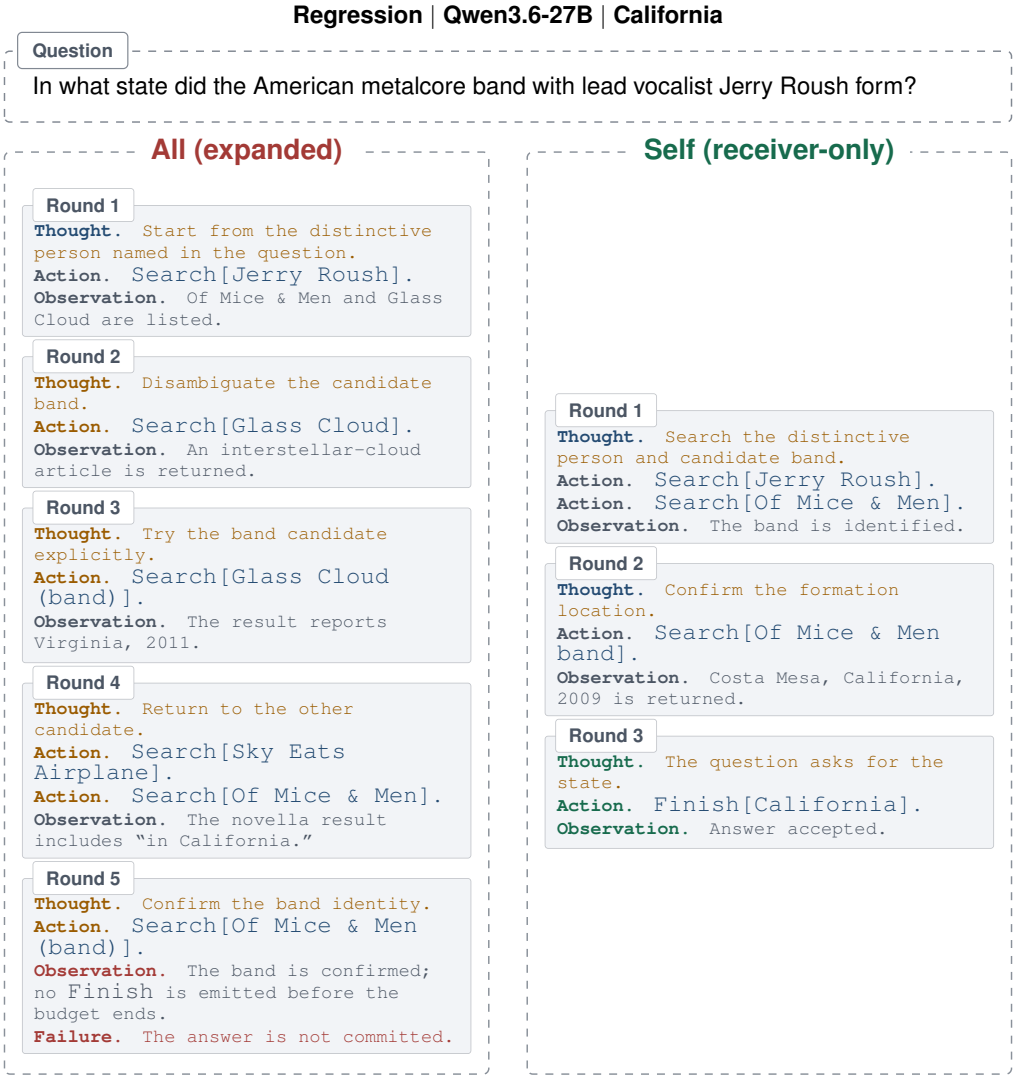
Round 5
Action. Search towelholder 1 and cabinets 1-3.
Observation. No cloth is found.

Round 6
Action. go to cabinet 4; open cabinet 4.
Observation. Cloth 1 is visible, but the run ends before pickup, cleaning, or placement.
Result. × Reward 0.

Our analysis finds that none of the seven added Peer Experience items addresses the error responsible for this failure. The injected Peer Experience nevertheless affects the model's behavior: compared with Self, All follows a different search and object-handling path, identifies the type error, and eventually acquires the actual cloth. However, this behavioral change does not help the model complete the held-out task, because the remaining action budget is insufficient to clean and place the cloth. The task therefore still fails despite the addition of multiple new experience items. The detailed manual judgments for these seven items are summarized in Table 10 below.

C.4.3 Regression: Added Evidence Induces Continued Search

The Qwen–HotpotQA case below illustrates a regression under source expansion. The task asks in which state an American metalcore band associated with vocalist Jerry Roush was formed. Under Self, Qwen identifies the target band, retrieves evidence that it formed in Costa Mesa, California, and submits the correct answer. Under All, Qwen explores several candidate bands and continues searching to resolve their identities, exhausting the interaction budget without submitting an answer. In this case, the expanded experience bank is associated with a longer search path and failure to terminate on a task that Self successfully completes.



The paired traces above show the reasoning and interactions under All and Self. Each interaction round separates the receiver's thought, emitted action, and resulting observation.

Specifically, All sees a search result containing "in California" at round 4, but this result describes the novella of the same name and does not establish where the target band was formed. At round 5, All continues with *Search[Of Mice & Men (band)]* and subsequently reaches the budget limit without a terminal action. By contrast, Self obtains the explicit band formation location, "Costa Mesa, California," at round 2 and emits *Finish[California]* at round 3. The regression in this case therefore reflects candidate-entity exploration that exhausts the interaction budget before the answer is submitted.

C.5 Absolute Scores for Fixed-Source Organization Controls

Table 11 reports the absolute scores underlying Figure 5(b). Within each benchmark–receiver setting, the source material, compiler, reconciliation procedure, and inference protocol remain fixed. Only the construction unit before reconciliation changes.

Table 11: Absolute scores for fixed-source experience-organization controls. Each row uses the same source material and inference protocol across the four construction units.

Benchmark	Fixed-source experience-organization controls			
	Self	RB-style	ExpeL-style	SkillGen-style
DeepSeek-V3.2				
ALFWorld	72.39	61.19	68.66	65.67
HotpotQA	46.00	45.00	38.00	36.00
WebShop	49.30	27.79	38.88	33.40
Qwen3.6-27B				
ALFWorld	88.06	86.57	89.55	90.30
HotpotQA	45.00	36.00	45.00	42.00
WebShop	54.40	55.76	55.00	49.33
GLM-5.1				
ALFWorld	91.04	85.82	90.30	88.81
HotpotQA	46.00	48.00	50.00	44.00
WebShop	55.50	50.66	51.33	48.87

C.6 Coverage-Gate Input-Token Audit

We estimate the input-evidence tokens used for Peer Experience extraction under Gated and Ungated admission using the same frozen source pool within each setting. Each fold-local success–failure pair is counted once. Table 12 reports the corresponding score, Peer-pair, and token-usage rows.

Table 12: Coverage-gated versus ungated All Experience. Each benchmark is split into score, Peer-pair, and estimated Peer-extraction token-usage rows. Within each model, columns are Gated, Ungated, and Δ =Gated−Ungated. Delta cells show the absolute difference as the primary value and the corresponding relative percentage in small type; positive score Δ favors gating, while the downward percentages on Peer-pair and token-usage rows report reductions under gating.

Benchmark	Measure	DeepSeek-V3.2			Qwen3.6-27B			GLM-5.1		
		Gated	Ungated	Δ	Gated	Ungated	Δ	Gated	Ungated	Δ
ALFWorld	Score	76.87	73.88	+2.99 +4.05%	91.79	94.03	−2.24 −2.38%	90.30	89.55	+0.75 +0.84%
	Peer pairs	246	421	−175 41.57%↓	69	158	−89 56.33%↓	54	95	−41 43.16%↓
	Token usage	211.0k	362.8k	−151.8k 41.84%↓	86.7k	186.2k	−99.5k 53.44%↓	74.2k	122.2k	−48.0k 39.24%↓
WebShop	Score	52.80	57.04	−4.24 −7.43%	59.03	58.20	+0.83 +1.43%	58.72	54.92	+3.79 +6.90%
	Peer pairs	117	315	−198 62.86%↓	36	132	−96 72.73%↓	117	168	−51 30.36%↓
	Token usage	96.7k	303.6k	−206.9k 68.16%↓	39.6k	151.6k	−112.0k 73.85%↓	95.3k	138.5k	−43.2k 31.24%↓
HotpotQA	Score	49.00	50.00	−1.00 −2.00%	46.00	45.00	+1.00 +2.22%	53.00	50.00	+3.00 +6.00%
	Peer pairs	162	354	−192 54.24%↓	225	315	−90 28.57%↓	108	198	−90 45.45%↓
	Token usage	210.2k	470.7k	−260.5k 55.35%↓	319.2k	441.1k	−121.9k 27.64%↓	161.5k	298.7k	−137.2k 45.92%↓

For setting i, the reduction is $r_i = 100(1 - G_i/U_i)$, where G_i and U_i are its Gated and Ungated estimates. We report the equally weighted arithmetic mean $\frac{1}{9}\sum_{i=1}^{9} r_i = 48.52\%$, rather than the ratio of pooled totals. This accounting includes only task text and paired trajectory evidence. It excludes the Self-experience construction cost common to both settings, fixed extraction instructions, the growing current-experience prefix, model outputs, abstraction, reconciliation, and retries.

D TASK-START TRACE PREPARATION AND EXPERIENCE–TRACE INTERACTION

This appendix supplements the analysis of trajectory cleaning and Experience–Trace interaction in Section 4.4. The paired Raw-to-Clean comparison holds the task, receiver, source trajectories, and task-start pairing fixed, changing only the injected trajectory representation. The component comparisons examine the contribution of trajectory references with All Experience held fixed, and the contribution of All Experience with trajectory references held fixed. The aggregate statistics and paired case studies below show how these changes affect task outcomes and execution behavior.

D.1 PAIRED TRACE-PREPARATION AUDIT

This section reports the aggregate evidence for the trace-cleaning results in Section 4.4. Table 6 reports the R=3 absolute performance of both trace-only conditions, and Table 13 summarizes the R1–R3 task-outcome flips and execution-action counts across all nine benchmark–receiver-model settings.

Across the full 3×3 audit, repaired episodes use fewer executed environment actions under Clean in seven of nine benchmark–receiver-model settings; HotpotQA–DeepSeek and WebShop–Qwen are the exceptions. Regression episodes use more executed environment actions in eight of nine settings; WebShop–GLM is the exception. The source-ID-aligned WebShop–GLM bank-fix runs provide the important counterexample: all 300 task-runs are fully aligned (100/100 in each replicate), yet both repairs and regressions become shorter under Clean. Thus, a shorter chain usually accompanies a positive flip but is not sufficient for a better task outcome. The matched task-level records are presented in Section D.2.

Table 13: Descriptive 3×3 Raw-to-Clean trace-preparation audit, aggregated over R1–R3. Raw actions and Clean actions report the mean executed environment-action counts for each condition; internal reasoning and actions blocked before environment execution are excluded, and counts are capped at the configured budget. Repairs and regressions are task-level success flips for ALFWorld and HotpotQA and binary buy-success flips for WebShop.

Benchmark	Repairs			Regressions		
	Repair	Raw	Clean	Regression	Raw	Clean
DeepSeek-V3.2						
ALFWorld	15	18.87	13.13	23	12.65	19.87
HotpotQA	26	5.62	5.69	28	5.64	6.00
WebShop	30	5.53	5.23	21	5.52	7.48
Qwen3.6-27B						
ALFWorld	17	20.00	11.41	35	10.57	12.23
HotpotQA	15	4.80	4.53	10	3.90	4.30
WebShop	26	4.50	5.12	20	4.55	7.00
GLM-5.1						
ALFWorld	3	20.00	17.33	7	14.14	20.00
HotpotQA	19	5.00	4.95	18	4.56	5.11
WebShop	14	8.64	6.29	15	5.27	3.07

D.2 TRAJECTORY-CLEANING CASE STUDIES

These cases provide task-level evidence for the two findings in Section 4.4: cleaned references can complement All Experience by supplying concrete execution paths, but cleaning alone does not yield consistent gains because different receiving tasks may require different decision information.

Table 14: R=3 matched component scores and effects. VetEx-Exp and Clean denote VetEx-Exp and Clean Trace-only, respectively. Scores are mean ± sample SD over three runs; each Δ is the VetEx mean minus the named configuration's mean, in percentage points.

Benchmark	VetEx	VetEx-Exp		Clean	
		Score	Δ	Score	Δ
DeepSeek-V3.2					
ALFWorld	90.55 ± 0.43	75.87 ± 1.17	+14.68	80.85 ± 0.86	+9.70
HotpotQA	48.67 ± 1.53	47.67 ± 1.15	+1.00	40.33 ± 3.06	+8.34
WebShop	60.47 ± 1.93	53.66 ± 1.25	+6.81	41.89 ± 1.69	+18.58
Qwen3.6-27B					
ALFWorld	92.04 ± 0.43	91.30 ± 0.87	+0.74	82.59 ± 1.14	+9.45
HotpotQA	47.33 ± 0.58	45.00 ± 3.61	+2.33	43.00 ± 0.00	+4.33
WebShop	64.72 ± 1.63	59.54 ± 0.59	+5.18	53.95 ± 0.99	+10.77
GLM-5.1					
ALFWorld	93.78 ± 1.87	90.80 ± 0.43	+2.98	89.80 ± 0.86	+3.98
HotpotQA	51.67 ± 2.08	52.33 ± 1.15	−0.66	48.67 ± 2.89	+3.00
WebShop	61.07 ± 0.79	60.45 ± 1.50	+0.62	57.02 ± 1.40	+4.05

D.2.1 Cleaning with All Experience

This subsection provides supplementary task-level behavioral evidence for the findings reported in Section 4.4. The two paired cases each control one component: the ALFWorld case holds All corrective experience fixed and compares whether a replay-validated, task-matched trajectory is added, whereas the HotpotQA case holds the task reference and source identities fixed and compares whether All corrective experience is added. The first case shows how task references provide a reusable concrete execution path; the second shows how corrective experience helps suppress unproductive search and terminate once the available evidence supports the answer.

ALFWorld prepared-path repair | Qwen3.6-27B

Question

Examine a book with a desk lamp; VetEx-Exp and VetEx use the same task, receiver, and 20-action budget; VetEx additionally receives the prepared task-start traces.

VetEx-Exp (0)

Round 1
Action. go to desk 1.
Observation. Desk 1 lists desklamp 1; no book.

Round 2
Action. go to desk 2.
Observation. Desk 2 lists an alarm clock, bowl, CD, mug, and pencil.

Round 3
Action. go to shelf 1.
Observation. Shelf 1 lists a bowl.

Round 4
Action. go to shelf 2.
Observation. Shelf 2 lists a credit card and pencil.

Round 5
Action. go to shelf 3.
Observation. Shelf 3 lists an alarm clock, bowls, and a desklamp.

VetEx (1)

Round 1
Action. go to desk 1.
Observation. Desk 1 lists desklamp 1; no book.

Round 2
Action. go to desk 2.
Observation. Desk 2 lists an alarm clock, bowl, CD, mug, and pencil.

Round 3
Action. go to shelf 1.
Observation. Shelf 1 lists a bowl.

Round 4
Action. go to shelf 2.
Observation. Shelf 2 lists a credit card and pencil.

Round 5
Action. go to shelf 3.
Observation. Shelf 3 lists an alarm clock, bowls, and a desklamp.

VetEx-Exp (0), continued

Round 6
Action. go to shelf 4.
Observation. Shelf 4 lists an alarm clock.

Round 7
Action. go to shelf 5.
Observation. Shelf 5 lists no objects.

Round 8
Action. go to shelf 6.
Observation. Shelf 6 lists no objects.

Round 9
Action. go to drawer 1.
Observation. Drawer 1 is closed.

Round 10
Action. open drawer 1.
Observation. Drawer 1 is open; no contents are listed.

Round 11
Action. go to drawer 2.
Observation. Drawer 2 lists cellphone 2.

Round 12
Action. go to drawer 3.
Observation. Nothing happens.

Round 13
Action. go to drawer 4.
Observation. Drawer 4 is closed.

Round 14
Action. open drawer 4.
Observation. Drawer 4 is open; no contents are listed.

Round 15
Action. go to drawer 5.
Observation. Drawer 5 lists no objects.

Round 16
Action. go to drawer 6.
Observation. Drawer 6 is closed.

Round 17
Action. open drawer 6.
Observation. Drawer 6 is open; no contents are listed.

Round 18
Action. go to bed 1.
Observation. Bed 1 lists book 1, phones, laptops, and pillows.

Round 19
Action. take book 1 from bed 1.
Observation. The book is picked up.

Round 20
Action. go to desk 1.
Observation. Desk 1 lists desklamp 1.
Result. × Reward 0.

VetEx (1), completed

Round 6
Action. go to shelf 4.
Observation. Shelf 4 lists an alarm clock.

Round 7
Action. go to shelf 5.
Observation. Shelf 5 lists no objects.

Round 8
Action. go to shelf 6.
Observation. Shelf 6 lists no objects.

Round 9
Action. go to drawer 1.
Observation. Drawer 1 is closed.

Round 10
Action. open drawer 1.
Observation. Drawer 1 is open; no contents are listed.

Round 11
Action. go to drawer 2.
Observation. Drawer 2 lists cellphone 2.

Round 12
Action. go to drawer 3.
Observation. Nothing happens.

Round 13
Action. go to bed 1.
Observation. Bed 1 lists book 1, phones, laptops, and pillows.

Round 14
Action. take book 1 from bed 1.
Observation. The book is picked up.

Round 15
Action. go to desk 1.
Observation. Desk 1 lists desklamp 1; the book is held.

Round 16
Action. use desklamp 1.
Observation. Task is SOLVED.
Result. ✓ Reward 1.

In the task *look_at_obj_in_light-Book-None-DeskLamp-308__33807892*, VetEx-Exp and VetEx share the All bank, receiver, temperature, environment, and the 20-action budget. VetEx additionally receives two task-type-matched compact traces. Both traces preserve the executable dependency *locate the target object → take it → return to the observed desk lamp → use the lamp*, ending with *use desklamp 1*. The target book is located from the live environment in both lanes; the compact references supply a reusable search–retrieval–return structure and preserve the final lamp-use operation. The added trace therefore contributes a concrete task-level action path, not a new task fact.



Both receivers obtain evidence that the house is in Hilo on the island of Hawaii. Clean Trace-only continues searching identifiers, registry records, and counties, and never emits *Finish*, even after identifying Hilo as the county seat of Hawaii County. VetEx preserves the evidence, performs one focused county check, and submits the correct answer with *Finish[Hawaii County]*. This paired contrast shows how All Experience guides the use of an existing trajectory reference: the receiver retains established evidence after a failed lookup, verifies unresolved information, and submits the answer once the evidence is sufficient. In this case, corrective experience constrains evidence use, verification scope, and termination timing, helping reduce unproductive search and turn the reference's execution guidance into task completion within budget.

D.2.2 CLEANING WITHOUT ALL EXPERIENCE

The Alice David task (*78d0f77ab8db*) and the Celene task (*d46838ef1661*) in HotpotQA provide a source-aligned comparison without corrective experience. Both use the same two source trajectories in the same order. Cleaning removes three search steps for one trajectory, while the other retains all four steps. Each step comprises a Thought–Action–Observation unit. In the paired rollouts shown below, the Alice David task changes from Raw failure to Clean success (EM: $0 \rightarrow 1$), whereas the Celene task changes from Raw success to Clean failure (EM: $1 \rightarrow 0$). We first show which steps cleaning removes from the source trajectory, then compare the Raw and Clean rollouts on both receiving tasks.



HotpotQA source trajectory: Raw/Clean comparison (continued)

Question

Kam Heskin plays Paige Morgan in a 2004 film directed by who?

Raw source

Source step 4

Thought. The search for "Paige Morgan 2004 film" gave results about a different actress (Paige Turco), not relevant. Let's search for "Kam Heskin 2004 film" to see her filmography that year.
Action. Search[Kam Heskin 2004 film]
Observation excerpt. ... and Paige Morgan in The Prince and Me film franchise (2006-2010).

Source step 5

Thought excerpt. ... Alternatively, maybe the film is "The Prince and Me" (2004) but the actress playing Paige Morgan in that film is actually Julia Stiles (who plays Paige Morgan), and Kam Heskin played a different role? ...
Action. Search[The Prince and Me 2004]
Observation excerpt. The Prince & Me is a 2004 American romantic comedy film directed by Martha Coolidge, and starring Julia Stiles, Luke Mably and Ben Miller, with Miranda Richardson, James Fox and Alberta Watson. ...

Source step 6

Thought. The search result clarifies that the original 2004 film "The Prince & Me" was directed by Martha Coolidge, and that Kam Heskin played Paige Morgan in the direct-to-video sequels (2006-2010), not in the 2004 film. However, the question asks: "Kam Heskin plays Paige Morgan in a 2004 film directed by who?" This seems contradictory because Kam Heskin did not play Paige Morgan in the 2004 film; Julia Stiles did. But perhaps the question is referring to one of the sequels that was released in 2004? The sequels started in 2006, so no. Maybe there is another 2004 film where Kam Heskin played a character named Paige Morgan. Let's search "Kam Heskin Paige Morgan 2004" more specifically.
Action. Search[Kam Heskin Paige Morgan 2004]
Observation excerpt. ... and Paige Morgan in The Prince and Me film franchise (2006-2010).

Source step 7

Thought. The search result for "The Prince and Me 2004" already gave the director as Martha Coolidge. Since the question might be referencing the franchise, the director of the original 2004 film is Martha Coolidge. I will answer with that.
Action. Finish[Martha Coolidge]
Observation. Answer is CORRECT

Clean source

Source step 4

×

Clean step 3

Thought excerpt. ... Perhaps the original 'The Prince & Me' film was released in 2004, and Heskin later took over the role in sequels. I should search for 'The Prince and Me 2004' to get details about the original film, including its director.
Action. Search[The Prince and Me 2004]
Observation excerpt. The Prince & Me is a 2004 American romantic comedy film directed by Martha Coolidge, and starring Julia Stiles, Luke Mably and Ben Miller, with Miranda Richardson, James Fox and Alberta Watson. ...

Source step 6

×

Clean step 4

Thought excerpt. ... Therefore, the 2004 film where the character Paige Morgan appears is 'The Prince & Me', and it was directed by Martha Coolidge. This answers the source task. I will finish with the director's name.
Action. Finish[Martha Coolidge]
Observation. Answer is CORRECT

The source trajectory contains seven bundled Thought–Action–Observation steps. Cleaning retains source steps 1, 2, 5, and 7. The deleted content is misleading in different ways: step 3 returns an unrelated biography of Paige Turco, while steps 4 and 6 repeat the Kam Heskin biography stating that her Paige Morgan role belongs to the 2006–2010 franchise, thereby keeping the apparent 2004/2006–2010 casting conflict active. By contrast, source step 5 provides the decisive evidence identifying Martha Coolidge as the director, and source step 7 does not repeat useless search but submits the answer.

HotpotQA target rollout: Alice David (Raw EM 0 → Clean EM 1)

Question

Alice David is the voice of Lara Croft in a video game developed by which company?

Raw Trace

Round 1
Action. Search[Alice David].
Observation. Alice David is the French voice of Lara Croft in the video game Tomb Raider.

Round 2
Action. Search[Tomb Raider French dub Alice David].
Observation. The same Alice David-Tomb Raider French-dub information is returned.

Round 3
Action. Search[Alice David Lara Croft Tomb Raider].
Observation. The result gives general information about Lara Croft and the Tomb Raider series.

Round 4
Action. Search[Tomb Raider 2013 French voice Alice David].
Observation. The Alice David-Tomb Raider French-dub information is returned again.

Round 5
Action. Search[Tomb Raider (2013 video game)].
Observation. The exact page is not found; similar Tomb Raider titles are returned.

Round 6
Action. Search[Tomb Raider 2013].
Observation. Tomb Raider is a 2013 game developed by Crystal Dynamics and published by Square Enix.

Round 7
Action. Lookup[French dub].
Observation. No Results.
Result. EM = 0.

Clean Trace

Round 1
Action. Search[Alice David].
Observation. Alice David is the French voice of Lara Croft in the video game Tomb Raider.

Round 2
Action. Search[Alice David Lara Croft].
Observation. The result gives general information about Lara Croft and the Tomb Raider series.

Round 3
Action. Search[Alice David Tomb Raider].
Observation. The Alice David-Tomb Raider French-dub information is returned.

Round 4
Action. Search[Tomb Raider 2013 French voice cast].
Observation. Tomb Raider is a 2013 game developed by Crystal Dynamics and published by Square Enix.

Round 5
Action. Lookup[French dub].
Observation. No Results.

Round 6
Action. Search["Alice David" "Lara Croft" voice].
Observation. The result returns Alice David's Lara Croft voice-acting information.

Round 7
Action. Finish[Crystal Dynamics].
Observation. Answer is correct.
Result. EM = 1.

HotpotQA target rollout: Celene (Raw EM 1 → Clean EM 0)

Question

The role of “Celene” in the film “Walk All over Me” was played by an actress that voices what role in the “StarCraft II” trilogy?

Raw Trace

Round 1
Action. Search[Walk All over Me].
Observation. Celene is played by Tricia Helfer.

Round 2
Action. Search[Tricia Helfer Starcraft 2].
Observation. Tricia Helfer information is returned, but the StarCraft role is not stated directly.

Round 3
Action. Lookup[Starcraft].
Observation. The result identifies Tricia Helfer as the voice of Sarah Kerrigan in StarCraft II.

Round 4
Action. Finish[Sarah Kerrigan].
Observation. Answer is CORRECT.
Result. EM = 1.

Clean Trace

Round 1
Action. Search[Walk All over Me].
Observation. Celene is played by Tricia Helfer.

Round 2
Action. Search[Tricia Helfer Starcraft 2].
Observation. Tricia Helfer information is returned, but the StarCraft role is not stated directly.

Round 3
Action. Lookup[voice].
Observation. An unrelated Battlestar Galactica voice-acting passage is returned.

Round 4
Action. Search[Tricia Helfer Starcraft].
Observation. General Tricia Helfer information is returned again.

Round 5
Action. Search[Starcraft 2 voice cast].
Observation. An unrelated HuskyStarcraft result is returned.

Round 6
Action. Search[StarCraft II voice actors Tricia Helfer].
Observation. General Tricia Helfer information is returned again.

Round 7
Action. Search[Sarah Kerrigan voice actor].
Observation. The result identifies Tricia Helfer as Sarah Kerrigan’s voice actor, but no answer is submitted.
Result. EM = 0.

On Alice David, both Actors initially follow a productive search path toward the correct answer. The first search identifies Alice David as the French voice of Lara Croft in *Tomb Raider*. The Raw Actor then repeats closely related French-dub searches at steps two and four, retrieves only general Lara Croft and Tomb Raider information at step three, and fails to find the exact page at step five. It reaches the decisive evidence only at step six, when the result identifies *Tomb Raider* as a 2013 game developed by Crystal Dynamics and published by Square Enix. Instead of submitting this answer, it uses the final step for another *Lookup[French dub]*, which returns no result. The rollout conditioned on the cleaned reference follows a shorter route: after several related searches, it obtains the Crystal Dynamics evidence at step four, uses the available information rather than continuing open-ended search, and executes *Finish[Crystal Dynamics]* at step seven. The difference is therefore a concrete change in behavior: the cleaned reference helps the Actor recognize that the current evidence supports the answer, whereas the uncleaned reference is associated with repeated search and a final diversion.

On Celene, the situation is reversed. Both Actors first identify the actress in *Walk All over Me* as Tricia Helfer, but their second search for *Tricia Helfer Starcraft 2* does not state her StarCraft role

directly. The Raw Actor then uses the broader *Lookup[Starcraft]*, which returns the decisive fact that Tricia Helfer voices Sarah Kerrigan in *StarCraft II*; it submits *Finish[Sarah Kerrigan]* at step four, leaving the remaining steps unused. The Clean Actor instead follows the unproductive cue in the question's wording and issues *Lookup[voice]*, which returns an unrelated Battlestar Galactica voice-acting passage. It then repeats general searches for Tricia Helfer, receives an unrelated HuskyStarcraft result from *Search[Starcraft 2 voice cast]*, and again obtains only general Tricia Helfer information from *Search[StarCraft II voice actors Tricia Helfer]*. Only at the final step does *Search[Sarah Kerrigan voice actor]* identify the correct role, leaving no remaining step in which to submit the answer. This trace shows why the exploratory and divergent behavior retained in the raw reference can help an Actor abandon an off-target search path, while the cleaned reference can leave the Actor following an unproductive local cue until its action budget is exhausted.

Taken together, the two traces show that the utility of information from the same source trajectories depends on the receiving Actor's current task state. In Alice David, the cleaned reference removes repeated and divergent searches, helping the Clean Actor recognize that the available evidence is sufficient and submit the answer, whereas the Raw Actor continues searching and fails to submit. In Celene, the exploratory and divergent behavior retained in the raw reference is useful because it helps the Raw Actor leave an off-target search path and identify Sarah Kerrigan, while the Clean Actor lacks this transition and follows unproductive local cues until its action budget is exhausted. Thus, the same source information can either mislead or constrain an Actor in one setting, or provide the corrective guidance needed to complete the task in another.

D.3 Relation between Compiler Capability and Experience Quality

This section provides experimental data for the relation examined in Section 4.5. We separately replace the compiler model with GLM-5.1, Qwen3.6-27B, and GPT-5.6-sol (high) for pairwise experience extraction and final reconciliation. All entries use the VetEx-E configuration. Table 15 reports the three complete WebShop replicates.

Table 15: WebShop compiler-backbone sensitivity. Scores are reward $\times 100$ and are reported as mean $\pm$ sample SD over three complete replicates.

Receiver model	**Compiler**		
	GLM-5.1	Qwen3.6-27B	GPT-5.6-sol (high)
DeepSeek-V3.2	53.66 ± 1.25	51.31 ± 1.60	55.22 ± 1.70
Qwen3.6-27B	59.54 ± 0.59	50.61 ± 1.88	51.44 ± 0.65
GLM-5.1	60.45 ± 1.50	59.71 ± 1.62	54.63 ± 1.97

D.4 Relation between Donor Capability and Trajectory Quality

This section provides experimental data for the relation examined in Section 4.5. We separately use Qwen3.6-27B and GPT-5.6-sol (high) as donor models supplying successful trajectories to the GLM-5.1 receiver. The evaluated configurations are VetEx-Exp, VetEx-Exp+RT, Clean Trace-only, and VetEx. Table 16 reports the results of three complete WebShop replicates for each configuration.

Table 16: WebShop donor-identity sensitivity with a fixed GLM-5.1 receiver model and compiler. Values are reward $\times 100$, mean $\pm$ sample SD over three replicates. The last column reports the difference between the GPT-5.6-sol (high) and Qwen donor means.

Configuration	Qwen donor	GPT-5.6-sol (high) donor	Δ GPT$-$Qwen
VetEx-Exp	60.45 ± 1.50	61.91 ± 3.57	$+1.46$
VetEx-Exp+RT	58.07 ± 1.27	61.51 ± 0.52	$+3.44$
Clean Trace-only	57.02 ± 1.40	55.08 ± 1.49	-1.94
VetEx	61.07 ± 0.79	61.94 ± 1.26	$+0.87$

E HISTORICAL EVIDENCE AND CONTEXT ACCOUNTING

Figure 7 reports per-benchmark averages over the available receiver-model reconstructions. Panel (b) measures the mean amount of experience information supplied by each method per task, reported in thousands of estimated tokens.

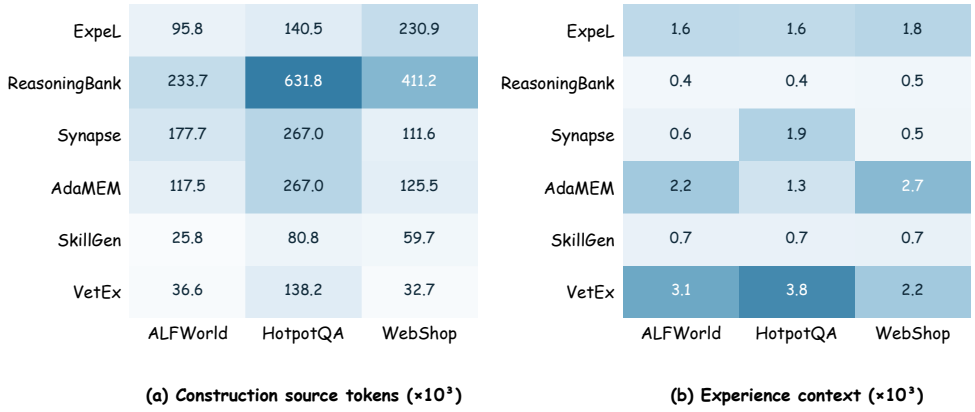


Figure 7: Per-benchmark context accounting shown as two independently scaled heatmaps. (a) Estimated source-trajectory input volume consumed during experience construction. (b) Mean experience information supplied per task, in estimated tokens.

E.1 CONSTRUCTION SOURCE TOKENS

For each test fold, we construct experience using only historical source trajectories from the other folds. We estimate the construction input volume by summing the estimated evidence-token lengths of the source trajectories actually consumed by the builder or indexer. These lengths are estimated using four characters per token.

E.2 EXPERIENCE CONTEXT

For methods whose task-level experience is fixed at task start, we distinguish two experience-reuse behaviors. Let $E^{\text{extract}}_{a,b,m,q}$ denote the corrective experience produced by experience extraction, and let $E^{\text{retrieve}}_{a,b,m,q}$ denote the successful-trajectory content provided by trajectory retrieval for task q. The estimated amount of experience information supplied for this task is

$$C_{a,b,m,q} = \frac{\text{chars}(E^{\text{extract}}_{a,b,m,q}) + \text{chars}(E^{\text{retrieve}}_{a,b,m,q})}{4}. \tag{1}$$

Here, $\text{chars}(\cdot)$ counts the characters in the corresponding experience text. We divide by four to place the text lengths on a common estimated-token scale.

We first average these task-level values over the $N_{a,b,m}$ available tasks for each receiver model,

$$\bar{C}_{a,b,m} = \frac{1}{N_{a,b,m}} \sum_{q=1}^{N_{a,b,m}} C_{a,b,m,q}, \tag{2}$$

and then compute an equally weighted mean over the available receiver models,

$$C_{a,b} = \frac{1}{M_{a,b}} \sum_{m \in \mathcal{M}_{a,b}} \bar{C}_{a,b,m}, \tag{3}$$

where $\mathcal{M}_{a,b}$ is the set of receiver models with available records and $M_{a,b} = |\mathcal{M}_{a,b}|$. Thus, each receiver model contributes the same weight to the benchmark-level estimate.

Methods with dynamically changing experience, such as AdaMEM, are evaluated with the same task-level and receiver-model averaging procedure. For task q, we sum the experience information newly supplied or refreshed during execution:

$$C_{a,b,m,q} = \sum_{r \in \mathcal{U}_{a,b,m,q}} \frac{\mathrm{chars}(E_{a,b,m,q,r})}{4}. \tag{4}$$

where $\mathcal{U}_{a,b,m,q}$ contains the distinct experience updates for that task. Repeatedly presenting unchanged experience does not create a new counted update. If an update contains multiple experience types, their character counts are added before division by four.

$$\mathrm{chars}(E_{a,b,m,q,r}) = \mathrm{chars}(E^{\mathrm{memory}}_{a,b,m,q,r}) + \mathrm{chars}(E^{\mathrm{strategy}}_{a,b,m,q,r}) + \mathrm{chars}(E^{\mathrm{reference}}_{a,b,m,q,r}). \tag{5}$$