CONTRAMEM: Learning Self-Evolving Procedural Memory from Contrasting Multi-Model Trajectories

Zheyuan Deng[1*†], Binghang Lu[2*], Hanqi Feng[3], Shirley Huang[4], Dianzhuo Wang[4]
Yuanda Xu[5], Zhiwei Zhang[6], Yige Sun[7], Changhong Mou[8], Runyu Zhang[9]
Yuexing Hao[9], Barnabas Poczos[3], Xiaomin Li[4†]

[1]Brown University [2]Purdue University [3]Carnegie Mellon University [4]Harvard University
[5]Princeton University [6]Pennsylvania State University [7]Independent Researcher
[8]Utah State University [9]Massachusetts Institute of Technology

Abstract

Autonomous computer-use agents are increasingly applied to long-horizon tasks requiring coordinated application calls, persistent state tracking, and verifier-sensitive writes, yet they remain prone to procedural failures—misreading application state, tool semantics, or task progress. Procedural memory promises more consistent decisions and less redundant exploration, but constructing high-quality memory without model training remains challenging. We introduce ContraMem, a source-flexible, training-free framework for self-evolving procedural memory that treats same-task outcome variation as supervision: differences in correctness, efficiency, recovery, and failure modes expose outcome-relevant procedural distinctions, distilled into a compact bank of app-level Function Cards and task-level Skill Cards that evolves through localized curation rather than append-only accumulation or whole-bank rewriting. On held-out GAIA2/ARE computer-use tasks, ContraMem more than doubles the success rate across the three source-model targets (26.2% to 55.3%), with consistent per-model gains (GPT-5.5: $27.5 \rightarrow 61.0$; Claude Sonnet 4.6: $28.0 \rightarrow 52.5$; DeepSeek V4 Pro: $23.0 \rightarrow 52.5$). The same bank transfers unchanged to the unseen Qwen3.7 Plus ($18.5 \rightarrow 35.5$), indicating transferable procedural knowledge rather than model-specific behavior. The same construction carries over unchanged to AppWorld, beating both no memory and its own single-source self-memory variant for all three mid-tier agents on both public test splits. Under a matched trajectory budget, heterogeneous multi-model trajectories yield stronger memory than self- or same-model multi-rollout memory: the margin comes from contrastive behavioral diversity, not stronger source agents or more sampling.

Introduction

Large language models are increasingly deployed as autonomous computer-use agents that interleave reasoning with action (Yao et al. 2022), invoke application functions and external tools (Schick et al. 2023; Patil et al. 2024; Li et al. 2023), and issue state-changing writes across realistic, multi-application environments (Lu et al. 2025; Trivedi et al. 2024; Froger et al. 2026). A successful agent in this regime must select the correct function, construct valid arguments, interpret asynchronous and often empty observations, maintain pending obligations, and stop only once the task is genuinely complete—and this difficulty does not disappear with scale: even strong models remain brittle on long-horizon tasks with state dependencies, temporal constraints, and verifier-sensitive writes (Trivedi et al. 2024; Froger et al. 2026). A natural, training-free way to close this gap is procedural memory: retained knowledge lets an agent avoid re-deriving equivalent searches, repeating the same function misuse, or relapsing into identical recovery failures (Shinn et al. 2023; Zhao et al. 2024; Wang et al. 2023b; Park et al. 2023; Packer et al. 2023).

Yet turning past experience into high-quality procedural memory without updating model weights is structurally difficult: raw retrieval transfers brittle surface detail instead of reusable logic, per-trajectory summaries miss the causal boundary separating success from failure, and same-model resampling suffers from *blind-spot inheritance*—a memory distilled from one policy reproduces the very gaps that caused its failures. Append-only accumulation compounds this with redundancy and drift: agents over-follow retrieved experience, propagate stale errors, and degrade as continuously updated memories accumulate faults (Fang et al. 2026; Xiong et al. 2025; Zhang et al. 2026; Xu et al. 2026; Zhang et al. 2025). The question is how to keep the bank compact, grounded, actionable, and retrievable as it evolves.

We identify heterogeneous exploration as an underused source of such memory—much as a team post-mortem that compares several attempts at the same incident teaches more than any single log. When distinct models attempt the same computer-use task, their behavior diverges informatively: one run succeeds where another fails, one path is markedly shorter, one model recovers where another stalls, or several fail at a common operation: same-task differences in correctness, efficiency, recovery, and failure that are not noise to be averaged away but supervision that multi-model practice already produces, at no extra labeling cost. Examining multiple paths surfaces stronger strategies than any single execution (Yao et al. 2023; Besta et al. 2024; DeepSeek-AI et al. 2025; Guo et al. 2026; Chang et al. 2026; Tang et al. 2025); for computer-use agents, such paths need not be consumed at test time and discarded. Instead, they can be distilled offline into reusable memory for future single-agent executions.

*These authors contributed equally.
†Corresponding author.

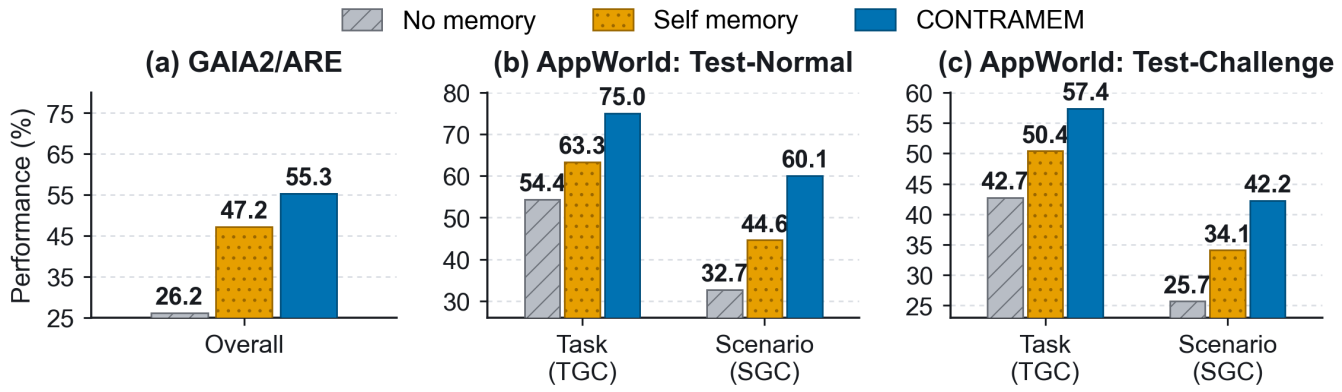

Figure 1: Held-out success averaged over three source targets (GAIA2/ARE) and target-macro TGC/SGC (AppWorld): ContraMem beats self memory on every metric.

Building on this view, we introduce ContraMem, a training-free, source-flexible framework that converts execution trajectories into a compact procedural memory bank. Given one or more trajectories for a reference task, ContraMem extracts outcome-relevant distinctions along four axes—correctness, efficiency, recovery, and recurring failure—and distills them, rather than the raw traces, into *Function Cards* (app-level tool contracts) and *Skill Cards* (task-level decision rules), maintained by a local Curator through small, targeted edits. Figure 2 summarizes this offline construction and the subsequent frozen-bank runtime. ContraMem does not require source diversity; it improves execution even from a single model's own trajectories. It reaches its strongest form, however, when sources are heterogeneous, transposing the central intuition of contrastive representation learning (van den Oord, Li, and Vinyals 2018; Chen et al. 2020; He et al. 2019) from representations to agent behavior: several models' attempts at one task are multiple views of its procedural structure, and their disagreements form a supervision signal that no single model can reliably supply, because same-model rollouts inherit one policy and one set of blind spots. The analogy is conceptual, not algorithmic: no embeddings, no contrastive objective. Multi-model exploration stays offline; deployment uses one target agent, and no parameters are updated.

We evaluate ContraMem on GAIA2/ARE, a dynamic app-agent benchmark with asynchronous state changes, ambiguity, time-sensitive obligations, and action-level write verification (Froger et al. 2026; Andrews et al. 2025), with GPT-5.5, Claude Sonnet 4.6, and DeepSeek V4 Pro as sources—a high-capability frontier where gains reflect headroom beyond raw capability. Across the three source target models, ContraMem more than doubles held-out success (26.2%→55.3%), transfers the bank unchanged to Qwen3.7 Plus, an unseen target ($18.5 \rightarrow 35.5$), and lands 30.0 points above the strongest prior memory system (Figure 3); Figure 1 summarizes this progression alongside AppWorld, where the same construction beats both no memory and target-specific self memory for all three mid-tier agents on both public test splits (Trivedi et al. 2024). Controlled ablations confirm the gains stem from cross-model heterogeneity rather than stronger sources or more sampling (Table 3).

Our contributions are threefold:

1. ContraMem: a source-flexible, training-free framework for self-evolving procedural memory, treating cross-model behavioral contrast as procedural supervision.
2. A granularity-aware representation pairing app-level Function Cards with task-level Skill Cards, maintained by a localized Curator through targeted edits.
3. Controlled evaluations that separately quantify the architecture's single-source gains and the added value of heterogeneous contrast, with transfer to an unseen target and cross-benchmark validation on AppWorld.

Related Work

Computer-use agents and app-agent benchmarks. Large language models have moved from text-only reasoning toward agents that act: ReAct introduced a general reasoning–acting loop, while Toolformer, Gorilla, and API-Bank studied how language models select and invoke external tools and APIs (Yao et al. 2022; Schick et al. 2023; Patil et al. 2024; Li et al. 2023). Recent benchmarks stress long-horizon, stateful app-agent behavior—conversational tool use in ToolSandbox, cross-application coding agents in AppWorld, and dynamic asynchronous tasks with ambiguity, time obligations, and action-level write verification in GAIA2/ARE (Lu et al. 2025; Trivedi et al. 2024; Froger et al. 2026)—exposing a gap ContraMem targets.

Trajectory-derived and experience-replay memory. A large body of work converts past executions into reusable experience: verbal reflections (Reflexion), lessons (ExpeL), executable skills (Voyager), induced workflows (Agent Workflow Memory) (Shinn et al. 2023; Zhao et al. 2024; Wang et al. 2023a, 2024), and distilled thought templates or procedural memories (Yang et al. 2024; Ouyang et al. 2026; Cao et al. 2026). Closer to our setting, Trajectory-Informed Memory Generation produces strategy and recovery tips for future AppWorld tasks (Fang et al. 2026), and Contextual Experience Replay keeps a dynamic buffer of environment dynamics for web agents (Liu et al. 2025). These methods,

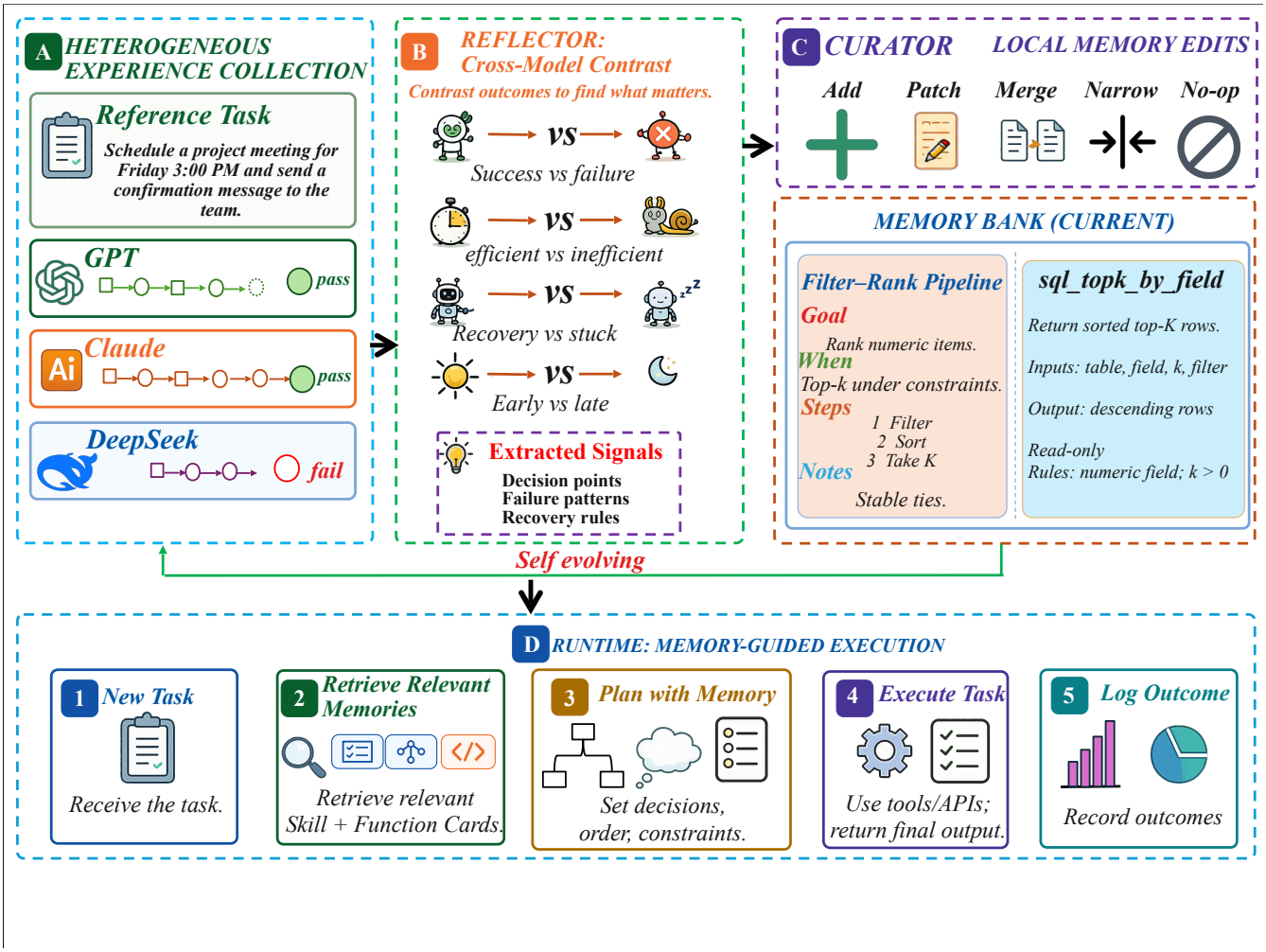

Figure 2: Overview of ContraMem: heterogeneous same-task trajectories (A) are compared by the Reflector to isolate outcome-relevant procedural differences (B); the Curator uses these signals to refine Function and Skill Cards in the current memory bank (C), from which a compact task-relevant subset guides a single target agent at runtime (D).

however, extract memory from individual trajectories or a single agent's experience, missing the contrastive boundary that explains why one execution succeeds while another fails. They also rarely separate function-level knowledge from task-level logic. ContraMem targets both gaps; its self-memory control instantiates this single-agent line inside the same architecture, isolating exactly what heterogeneous contrast adds.

Evolving contexts and memory management. A parallel line emphasizes that memory quality depends on maintenance (Lu et al. 2026). A-MEM organizes agent memories as structured, linked notes that evolve as new memories arrive (Xu et al. 2026); ACE treats context as an evolving playbook and argues that whole-context rewriting causes brevity bias and context collapse, motivating incremental generation–reflection–curation updates (Zhang et al. 2025); and empirical studies confirm these failure modes in continuously updated memories (Xiong et al. 2025; Zhang et al. 2026). In its released offline path, ACE builds a single playbook from an agent's own executions and injects it in full; ContraMem instead derives typed cards from same-task cross-model contrasts, refines them through localized curation, and retrieves only task-relevant guidance.

Cross-agent, cross-model, and multi-path learning. A final line studies how multiple solution paths or agents improve reasoning and reuse: Tree and Graph of Thoughts explore multiple reasoning paths (Yao et al. 2023; Besta et al. 2024), SE-Agent evolves trajectories by revision and recombination (Guo et al. 2026), and Agent KB aggregates cross-framework trajectories (Tang et al. 2025). Most directly related is MemCollab, which likewise contrasts trajectories from different models on the same task (Chang et al. 2026)—but toward transferability, distilling agent-invariant constraints so one memory can be shared across backbones. ContraMem uses the same disagreement for a different end: rather than normalizing model differences away, it exploits

them to raise memory quality among frontier sources under a matched budget.

Methodology

Problem Setup

ContraMem constructs a typed memory bank from reference-task trajectories and retrieves a small task-relevant subset for held-out execution.

With $\mathcal{A}$ the ability set, tasks in each $a \in \mathcal{A}$ are split into a reference set $\mathcal{D}^a_{\text{ref}}$ and a held-out set $\mathcal{D}^a_{\text{test}}$; each source agent $m \in \mathcal{M}$ attempts each reference task x_i, producing a trajectory $\tau_{i,m} = (e_1, \ldots, e_{T_{i,m}}, y_{i,m})$ of interaction events with a verifier-assigned outcome. From these, ContraMem constructs a bank $\mathcal{B} = (\mathcal{F}, \{\mathcal{S}^a\})$: a global Function Card bank $\mathcal{F}$ shared across abilities and ability-specific Skill Card banks $\mathcal{S}^a$. At evaluation time, a target agent g, possibly $g \notin \mathcal{M}$, receives guidance rendered by the deterministic retriever $\rho(x, \mathcal{B})$ while attempting $\mathcal{D}^a_{\text{test}}$; no model parameters are updated. Algorithms 1–2 in Appendix A.8 summarize the complete construction and runtime pipeline.

This separates two transfers: Function memory is global because a function keeps its semantics across task families; Skill memory stays ability-specific because one surface pattern implies different obligations—a message write may be a final answer in search, a side effect in execution, or a clarification boundary in ambiguity.

Trajectory normalization. Raw trajectories interleave model messages, app calls and observations, environment notifications, verifier feedback, and final responses. We map each trace to a common event sequence recording source, type, and position and, for app calls, the function, normalized arguments, and a compact observation summary. Agent-callable functions ground Function Cards; environment and verifier events are never tools but stay available for Skill reflection. Before any artifact reaches a construction model, task-specific values are replaced under the placeholder policy (Appendices B.3–B.5) while dates, times, and operation order are retained.

Two intermediate artifacts derive from this sequence: a *function observation* captures one callable function's arguments, return form, side effects, and trajectory outcome, while a *task contrast packet* groups one task's source trajectories—task text, outcomes, ordered action spans, writes, final responses, and normalized verifier feedback—exposing where runs diverge while preserving the read-/write/wait/finalize order that determined each outcome.

Function Card Construction

Each Function Card compactly describes one callable app function; because a function's interface and side effects are shared across tasks, its card is reused across abilities. For every observed function u, a builder condenses its calls from all reference trajectories (evidence about accepted arguments, return forms, empty or error cases, and state-changing effects) into one card f_u: what it does, how to call it safely, what it returns, and which mistakes need caution. Cards remain strictly tool-level: every claim must be supported by an observed call, and no card may contain a task plan, trajectory summary, model identity, or verifier judgment. This keeps them useful both at task start and immediately before the function is called (schema in Appendix B.1, builder prompt in Appendix B.3, example in the left plate of Appendix Figure B1).

Contrastive Skill Memory

Skill Cards encode transferable procedural rules above individual function syntax. For each reference task, ContraMem compares the source trajectories through the contrast packet—not to summarize every run, but to locate the decision boundary that explains the outcome contrast. For example, when failing agents guess an underspecified variant and execute a write while a successful agent completes the safe branch and asks, the resulting card blocks the ambiguous write until clarification. Supplementary Figure B2 traces this rule from cross-model disagreement to successful reuse on a new task; rendered card formats appear in Supplementary Figure B1.

Reflector. Given a contrast packet, the relevant current cards, and an ability-specific focus, the Reflector isolates the decision boundary where same-task trajectories diverged, attributes the success, failure, recovery, or efficiency contrast, and abstracts it one level above the scenario into a reusable rule with an explicit completion condition. Two constraints keep the output grounded: every field must be supported by packet evidence, and if all source trajectories fail, the Reflector may emit only diagnostic or recovery deltas rather than inventing a success recipe (full protocol and focus blocks in Appendix B.4).

The Reflector returns at most three candidate deltas per the Skill Delta schema in Appendix B.1; the right plate of Appendix Figure B1 shows a rendered card.

Curator. Candidate deltas are not written directly into the bank: for each delta, the Curator (Appendix B.5) retrieves the most relevant existing cards from the same ability bank and decides the smallest valid edit among `ADD`, `PATCH`, `MERGE`, `NARROW`, and `NOOP` (patch schema in Appendix B.1). The Curator preserves semantic closure: it may combine, compress, narrow, or rephrase information supported by the delta, the targeted cards, or related Function Cards, but can introduce no new rule, failure mode, example, function, or completion condition—preventing drift toward generic advice and narrow-evidence rewrites.

The resulting Skill Card keeps the delta's compact runtime fields; provenance, bookkeeping artifacts, and curator reasoning remain outside runtime memory. The five final ability banks contain 256 Skill Cards; across them the Curator makes 394 localized decisions: 262 additions and 132 patches, merges, trigger narrowings, or no-ops (Appendix Figure A1). Thus bank growth is governed by evidence-sensitive local edits rather than append-only accumulation.

Retrieval and Runtime Injection

At evaluation time, the task's ability label selects which skill bank is eligible; when no label is available (as in open-ended

deployment), retrieval simply retrieve over all banks. Within the selected bank, the reported GAIA2/ARE retriever ranks active Skill Cards by BM25 relevance with a capped soft app-domain overlap bonus and a domain-mismatch penalty. These domain terms affect ranking rather than hard-gating cross-family transfer, so app-independent procedures, such as requesting clarification before an ambiguous write, remain retrievable across application families. Appendix A.8 gives the frozen benchmark-specific scores and selection caps. Function Cards follow a separate path: the top globally relevant cards are injected at task start; when the runtime identifies a proposed call, the matching card surfaces immediately before it as a just-in-time reminder of contract, returns, side effects, and common mistakes.

Injected memory is soft procedural guidance: current app observations remain ground truth, and the target must not copy stored entities, dates, identifiers, or final answers (complete task-start and pre-tool templates in Appendix B.6).

Validation. ContraMem validates memory at three levels: schema validation rejects any card or patch unsupported by the delta, the targeted cards, or related Function Cards; privacy validation rejects contact details, long identifiers, raw judge prose, and copied scenario-specific values; and retrieval preview verifies that held-out queries retrieve plausible cards from the correct bank without same-scenario leakage. Without these checks, a bank can look strong by memorizing tasks or replaying values.

Experiments

Experimental Setup

We evaluate ContraMem on GAIA2/ARE, the GAIA2 benchmark executed in the Agents Research Environments (ARE) runtime (Froger et al. 2026; Andrews et al. 2025), across five abilities (Execution, Search, Ambiguity, Adaptability, Time), with 40 reference and 40 held-out tasks per ability. The bank is built from three heterogeneous source models (GPT-5.5, Claude Sonnet 4.6, and DeepSeek V4 Pro) and evaluated on these target families plus Qwen3.7 Plus, reserved as the unseen-transfer target with no contributed trajectories. Each source target also gets a self-memory control: the same pipeline on only its own reference trajectories. All agents are frontier-level, so gains are headroom on top of already-capable agents. Construction consumes 600 offline source trajectories (5 abilities $\times$ 40 tasks $\times$ 3 models) and yields a frozen bank of 256 Skill Cards and 74 global Function Cards (Appendix Figure A1). All experiments use the default runtime except the Time ability, where wrapper-timestamp confounds require the normalized runtime for all conditions. ContraMem retrieves up to three Skill Cards at start of the task and supplies Function Card guidance before identifiable tool calls; every main evaluation compares the three conditions on the same split (frozen configuration: Appendix Table A1). Our primary metric is task success rate (the fraction of tasks passing the environment verifier); efficiency is measured in *agent events*, the number of agent-level trajectory events per scenario.

To compare against published agent-memory baselines, we evaluate offline Agent Workflow Memory (AWM) (Wang et al. 2024), ACE (Zhang et al. 2025), and raw trajectory retrieval on GPT-5.5 over Execution, Search, and Ambiguity, testing whether the gains reduce to trajectory replay, induced workflows, or an evolving playbook. All conditions share the three-model source pool and held-out scenarios (adaptation and accounting details in Appendices A.6–A.7).

Main Held-Out Results

Table 1 shows that ContraMem more than doubles held-out success on tasks unseen during construction, raising the source-target macro from 26.2% to 55.3% with consistent per-target gains (GPT-5.5 +33.5, Claude Sonnet +24.5, DeepSeek V4 Pro +29.5). Self memory, which runs the same pipeline on one model's trajectories, is already effective (47.2% source-target macro versus 26.2% without), and the shared bank adds 8.2 points on top, reaching 55.3% and transferring to the unseen Qwen3.7 Plus ($18.5 \rightarrow 35.5$): ContraMem distills reusable guidance, not model-specific behavior. All four non-temporal abilities improve (Execution +33.1, Search +31.2, Ambiguity +34.4, Adaptability +29.4) with Analysis tracing the gains to the memory contents.

Time ability remains hardest: obligations must survive asynchronous notifications and land within narrow verifier windows against a simulated clock. Memory still helps behaviorally (GPT-5.5 rises from $3/40$ to $7/40$; hung runs drop sharply, Appendix A.2), but self and contrastive memory remain close across the three source targets ($10/120$ versus $9/120$): cross-model evidence alone does not resolve temporal control. This marks the limit of text-injected procedural memory: it reliably prevents the failures it has encoded; errors beyond the source evidence call for a runtime controller, not richer recall.

Across all paired held-out runs, the gains are statistically significant (232 failure-to-pass versus 23 pass-to-fail flips; McNemar $\chi^2 = 171.3$, $p \ll 0.001$) (McNemar 1947). Across all five abilities, ContraMem reduces the macro-average trajectory length from 41.1 to 35.5 agent events per task (-5.6 events; -13.6%).

Cross-Benchmark Validation on AppWorld

We further validate the same recipe on AppWorld (Trivedi et al. 2024), which changes nearly everything about the execution regime: the agent writes Python in a stateful REPL—orchestrating roughly 450 APIs across nine applications, consulting live documentation, and finalizing through an explicit completion call. State-based unit tests check required effects and collateral damage; results are task (TGC) and scenario (SGC) goal completion on both public splits (Test-Normal: 168 tasks / 56 scenarios; Test-Challenge: 417 / 139).

Because frontier models leave little headroom on much of AppWorld, we deliberately evaluate three mid-tier agents, including DeepSeek-V3.1, Qwen3.6-Flash, and GPT-4.1-mini, so measurement again reflects procedural headroom, not raw capability. The same three agents serve as sources, each attempting every training-split task once; the same-task triples are contrasted exactly as on GAIA2. Two construction elements are benchmark-aware rather than benchmark-tuned. Because AppWorld agents can read live documentation at any point, Function/API Cards obey a *doc-delta* rule: they

Target	Method	Exec.	Search	Ambig.	Adapt.	Time	Overall
Source target models							
GPT-5.5	No memory	47.5	52.5	12.5	17.5	7.5	27.5
	Self memory	72.5	92.5	40.0	35.0	12.5	50.5 (+23.0)
	ContraMem	**80.0**	**100.0**	**52.5**	**55.0**	**17.5**	**61.0** (+33.5)
Claude Sonnet	No memory	40.0	57.5	17.5	22.5	2.5	28.0
	Self memory	60.0	80.0	52.5	**52.5**	**10.0**	51.0 (+23.0)
	ContraMem	**75.0**	**85.0**	**57.5**	42.5	2.5	**52.5** (+24.5)
DeepSeek V4 Pro	No memory	35.0	47.5	12.5	17.5	2.5	23.0
	Self memory	52.5	**77.5**	32.5	35.0	2.5	40.0 (+17.0)
	ContraMem	**75.0**	**77.5**	**55.0**	**52.5**	2.5	**52.5** (+29.5)
Source macro	No memory	40.8	52.5	14.2	19.2	4.2	26.2
	Self memory	61.7	83.3	41.7	40.8	**8.3**	47.2 (+21.0)
	ContraMem	**76.7**	**87.5**	**55.0**	**50.0**	7.5	**55.3** (+29.2)
Unseen target model							
Qwen3.7 Plus	No memory	25.0	47.5	5.0	15.0	0.0	18.5
	ContraMem	**50.0**	**67.5**	**20.0**	**40.0**	0.0	**35.5** (+17.0)

Table 1: Held-out success rates (%) on GAIA2/ARE (40 tasks per ability and target). Self memory applies the same pipeline to only the target's own reference trajectories; ContraMem uses the shared three-model bank. Qwen3.7 Plus is unseen during construction, hence no self-memory row. Bold: best per column; parentheses: absolute gain over no memory.

record only evidence-backed behavior beyond the documentation, such as exception signatures, argument constraints, and pagination, and never restate it. In addition, a deterministic task signature (answer extraction, bulk/multi-write, state write, cross-app, default) selects the reflection focus, replacing GAIA2's ability focus. Construction uses GPT-4.1-mini, the weakest of the trio, as Reflector and Curator, so gains cannot come from a stronger construction model; each self-memory control is fully self-contained, the target curating its own trajectories. All banks are frozen; retrieved cards (at most three Skill and two Function/API Cards, about 1.1k tokens) are injected once at task start as soft guidance under an unchanged prompt. Construction yields 32 Skill and 24 Function/API Cards over 60 Curator decisions (Appendix Figure A1; prompt suite and injection contract in Appendices B.7–B.11).

Table 2 shows a consistent ordering across every target, split, and metric: ContraMem outperforms both no memory and target-specific self memory. Averaged over the three targets, Test-Normal TGC rises from 54.4% without memory to 63.3% with self memory and 75.0% with ContraMem; on Test-Challenge the progression is $42.7\% \rightarrow 50.4\% \rightarrow 57.4\%$, with per-target gains of $+16.7$–$+25.0$ (Test-Normal) and $+13.2$–$+17.5$ (Test-Challenge). The pattern is strongest for SGC, which requires every task in a scenario to pass: the cross-model bank reaches 60.1% and 42.2% versus 44.6% and 34.1% for self memory—informative, because scenario completion rewards exactly the discipline (complete enumeration before writing, verification before finalization) the cards encode. The consistent margin over self memory shows that heterogeneous trajectories add value beyond retention, without ensembling or parameter updates.

Ablations and Analysis

We run cost-controlled ablations isolating source diversity, curation, and memory type on the three most diagnostic abilities (Search, Ambiguity, and Execution) on the held-out split with the frozen retrieval path. Table 3 reports source and component ablations; Figure 3 compares prior memory systems.

Source Diversity and Curation

Self memory is not a third-party baseline but ContraMem run single-source, with the same normalization, typed cards, Curator, and retrieval; its 26.2%→47.2% source-target gain (Table 1) measures the architecture alone; the shared bank's 55.3% (margins +10.5 on GPT-5.5, +12.5 on DeepSeek) isolates the added value of heterogeneous contrast. To separate diversity from trajectory count, Table 3 adds a matched multi-rollout control with three GPT-5.5 trajectories per task; it trails ContraMem on every ability and on macro average (70.0 vs. 77.5): additional sampling alone does not explain the gain.

We isolate the Curator against an append-only variant committing every Reflector delta as Add, with identical deltas and retrieval. On the four non-temporal abilities covered by this ablation, local curation removes 24.9% of Skill Cards ($273 \rightarrow 205$) while improving GPT-5.5 held-out macro success from 65.0% to 71.9% (Appendix Table A4)—a smaller bank, no ability regressing, strongest on Execution (+15.0), where redundant write-coverage cards compete for fixed injection slots.

Memory Types and Prior Memory Systems

The single-memory-type rows of Table 3 isolate the two levels: Skill Cards carry much of Search and Ambiguity, Function Cards matter most where correctness depends on

Target	Method	Test-Normal TGC	Test-Normal SGC	Test-Normal Δ Base	Test-Challenge TGC	Test-Challenge SGC	Test-Challenge Δ Base
DeepSeek-V3.1	No memory	51.2	30.4	–	56.8	38.1	–
	Self memory	56.0	35.7	+4.8	64.3	46.8	+7.4
	ContraMem	**76.2**	**58.9**	**+25.0**	**70.0**	**56.1**	**+13.2**
Qwen3.6-Flash	No memory	63.7	42.9	–	46.0	31.7	–
	Self memory	73.2	58.9	+9.5	50.8	37.4	+4.8
	ContraMem	**80.4**	**71.4**	**+16.7**	**59.5**	**47.5**	**+13.4**
GPT-4.1-mini	No memory	48.2	25.0	–	25.2	7.2	–
	Self memory	60.7	39.3	+12.5	36.2	18.0	+11.0
	ContraMem	**68.5**	**50.0**	**+20.2**	**42.7**	**23.0**	**+17.5**
Target macro	No memory	54.4	32.7	–	42.7	25.7	–
	Self memory	63.3	44.6	+8.9	50.4	34.1	+7.8
	ContraMem	**75.0**	**60.1**	**+20.6**	**57.4**	**42.2**	**+14.7**

Table 2: AppWorld task (TGC) and scenario (SGC) goal completion (%) on both public test splits. Self memory applies the pipeline to each target's own train-split trajectories (self-curated); ContraMem uses the shared three-model bank; all memories are frozen. Bold: best. Δ Base: TGC gain over no memory.

Memory variant	Exec.	Search	Ambig.	Macro
No memory	47.5	52.5	12.5	37.5
GPT-5.5 self memory	72.5	92.5	40.0	68.3
GPT-5.5 multi-rollout	75.0	92.5	42.5	70.0
Function Cards only	67.5	85.0	30.0	60.8
Skill Cards only	60.0	100.0	40.0	66.7
ContraMem (three models)	**80.0**	**100.0**	**52.5**	**77.5**

Table 3: GPT-5.5 held-out source and component ablations. Multi-rollout uses three GPT-5.5 trajectories, matching ContraMem's count; single-card rows use the shared bank and frozen retrieval path. Macro averages the three abilities.

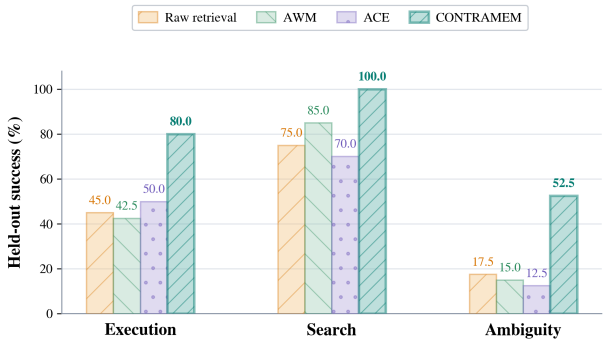

Figure 3: ContraMem versus trajectory- and playbook-based memory on GPT-5.5 held-out tasks. All methods share the same source pool and scenarios. Exact values: Appendix Table A5.

concrete writes (Execution), and their combination attains the best macro average. The levels are complementary.

Figure 3 compares ContraMem with raw retrieval, offline AWM (Wang et al. 2024), and ACE (Zhang et al. 2025) under their native learning rules. AWM induces a fixed workflow library from verifier-passing trajectories, whereas ACE turns all 120 trajectories per ability into an evolving full playbook; neither uses same-task contrasts, typed Function/Skill memory, or task-conditioned retrieval (Appendices A.6–A.7). AWM is the strongest prior baseline (47.5% macro; 17 vs. 5 discordant flips, p=0.0169), followed by raw retrieval (45.8%) and ACE (44.2%). ContraMem reaches 77.5%: 30.0 points above AWM, with 38 favorable flips against 2 ($p<10^{-8}$), showing that preserving decision boundaries is more useful than retaining trace detail alone.

Analysis. The largest gains come where the no-memory agent has enough evidence but writes, asks, retries, or stops at the wrong boundary: in Ambiguity, retrieved skills convert "guess and write" into "complete safe branches and ask before side effects" (Supplementary Figure B2); in Search, they identify when the answer is determined; in Execution and Adaptability, they preserve obligations. The 23 pass-to-fail regressions arise mainly from over-broad clarification cards or caution around already-determined writes, motivating conservative top-k retrieval. Transfer to Qwen3.7 Plus (18.5 → 35.5, with no contributed trajectories) further indicates that shared-scenario divergence exposes task- and tool-level invariants rather than source-specific detours.

Conclusion

We introduced ContraMem, which turns same-task disagreement into procedural supervision and distills it into a compact, locally curated bank of Function and Skill Cards. It more than doubles held-out success rate on GAIA2/ARE, outperforms single-source, workflow, and playbook memory, transfers unchanged to an unseen target, and improves three target agents on both AppWorld test splits.

Our work admittedly has limitations: ContraMem remains bounded by the procedural failures represented in its source trajectories, and text-injected memory alone does not

resolve time-sensitive failures requiring persistent state tracking and runtime control. Combining curated memory with mechanisms for obligation tracking and asynchronous-state monitoring is a promising next step.

References

Andrews, P.; Benhalloum, A.; Bertran, G. M.-T.; et al. 2025. ARE: Scaling Up Agent Environments and Evaluations. *arXiv preprint arXiv:2509.17158*.

Besta, M.; Blach, N.; Kubicek, A.; Gerstenberger, R.; Podstawski, M.; Gianinazzi, L.; Gajda, J.; Lehmann, T.; Niewiadomski, H.; Nyczyk, P.; et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, 17682–17690.

Cao, Z.; Deng, J.; Yu, L.; Zhou, W.; Liu, Z.; Ding, B.; and Zhao, H. 2026. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. In *Findings of the Association for Computational Linguistics: ACL 2026*, 16803–16822.

Chang, Y.; Wu, Y.; Wu, Q.; and Lin, L. 2026. MemCollab: Cross-Agent Memory Collaboration via Contrastive Trajectory Distillation. *arXiv preprint arXiv:2603.23234*.

Chen, T.; Kornblith, S.; Norouzi, M.; and Hinton, G. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, 1597–1607. PMLR.

DeepSeek-AI; Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948*.

Fang, G.; Isahagian, V.; Jayaram, K. R.; Kumar, R.; Muthusamy, V.; Oum, P.; and Thomas, G. 2026. Trajectory-Informed Memory Generation for Self-Improving Agent Systems. *arXiv preprint arXiv:2603.10600*.

Froger, R.; Andrews, P.; Bettini, M.; Budhiraja, A.; Cabral, R. S.; Do, V.; Garreau, E.; Gaya, J.-B.; Laurençon, H.; Lecanu, M.; Malkan, K.; Mekala, D.; Ménard, P.; Bertran, G. M.-T.; Piterbarg, U.; Plekhanov, M.; Rita, M.; Rusakov, A.; Vorotilov, V.; Wang, M.; Yu, I.; Benhalloum, A.; Mialon, G.; and Scialom, T. 2026. Gaia2: Benchmarking LLM Agents on Dynamic and Asynchronous Environments. *arXiv preprint arXiv:2602.11964*.

Guo, Y.; Lin, J.; Wang, H.; Han, Y.; Hu, S.; Ni, Z.; Wang, L.; and Chen, M. 2026. SE-agent: Self-evolution trajectory optimization in multi-step reasoning with LLM-based agents. *Advances in Neural Information Processing Systems*, 38: 116314–116341.

He, K.; Fan, H.; Wu, Y.; Xie, S.; and Girshick, R. 2019. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*.

Li, M.; Zhao, Y.; Yu, B.; Song, F.; Li, H.; Yu, H.; Li, Z.; Huang, F.; and Li, Y. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, 3102–3116.

Liu, Y.; Si, C.; Narasimhan, K. R.; and Yao, S. 2025. 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)*, 14179–14198.

Lu, B.; Deng, Z.; Zhang, R.; Hu, B.; Zhao, Y.; Tian, Y.; Mou, C.; Lin, G.; and Li, X. 2026. Muon-OGD: Muon-based spectral orthogonal gradient projection for LLM continual learning. *arXiv preprint arXiv:2605.08949*.

Lu, J.; Holleis, T.; Zhang, Y.; Aumayer, B.; Nan, F.; Bai, H.; Ma, S.; Ma, S.; Li, M.; Yin, G.; et al. 2025. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, 1160–1183.

McNemar, Q. 1947. Note on the Sampling Error of the Difference Between Correlated Proportions or Percentages. *Psychometrika*, 12(2): 153–157.

Ouyang, S.; Yan, J.; Hsu, I.-H.; Chen, Y.; Jiang, K.; Wang, Z.; Han, R.; Le, L. T.; Daruki, S.; Tang, X.; Tirumalashetty, V.; Lee, G.; Rofouei, M.; Lin, H.; Han, J.; Lee, C.-Y.; and Pfister, T. 2026. ReasoningBank: Scaling Agent Self-Evolving with Reasoning Memory. *arXiv preprint arXiv:2509.25140*.

Packer, C.; Wooders, S.; Lin, K.; Fang, V.; Patil, S. G.; Stoica, I.; and Gonzalez, J. E. 2023. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560*.

Park, J. S.; O'Brien, J.; Cai, C. J.; Morris, M. R.; Liang, P.; and Bernstein, M. S. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, 1–22.

Patil, S. G.; Zhang, T.; Wang, X.; and Gonzalez, J. E. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37: 126544–126565.

Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Hambro, E.; Zettlemoyer, L.; Cancedda, N.; and Scialom, T. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36: 68539–68551.

Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K.; and Yao, S. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36: 8634–8652.

Tang, X.; Qin, T.; Peng, T.; Zhou, Z.; Shao, D.; Du, T.; Wei, X.; Xia, P.; Wu, F.; Zhu, H.; Zhang, G.; Liu, J.; Wang, X.; Hong, S.; Wu, C.; Cheng, H.; Wang, C.; and Zhou, W. 2025. Agent KB: Leveraging Cross-Domain Experience for Agentic Problem Solving. *arXiv preprint arXiv:2507.06229*.

Trivedi, H.; Khot, T.; Hartmann, M.; Manku, R.; Dong, V.; Li, E.; Gupta, S.; Sabharwal, A.; and Balasubramanian, N. 2024. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 16022–16076.

van den Oord, A.; Li, Y.; and Vinyals, O. 2018. Representation Learning with Contrastive Predictive Coding. *arXiv preprint arXiv:1807.03748*.

Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2023a. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv preprint arXiv:2305.16291*.

Wang, W.; Dong, L.; Cheng, H.; Liu, X.; Yan, X.; Gao, J.; and Wei, F. 2023b. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36: 74530–74543.

Wang, Z. Z.; Mao, J.; Fried, D.; and Neubig, G. 2024. Agent Workflow Memory. *arXiv preprint arXiv:2409.07429*.

Xiong, Z.; Lin, Y.; Xie, W.; He, P.; Tang, J.; Lakkaraju, H.; and Xiang, Z. 2025. How Memory Management Impacts LLM Agents: An Empirical Study of Experience-Following Behavior. *arXiv preprint arXiv:2505.16067*.

Xu, W.; Liang, Z.; Mei, K.; Gao, H.; Tan, J.; and Zhang, Y. 2026. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38: 17577–17604.

Yang, L.; Yu, Z.; Zhang, T.; Cao, S.; Xu, M.; Zhang, W.; Gonzalez, J. E.; and Cui, B. 2024. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems*, 37: 113519–113544.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36: 11809–11822.

Yao, S.; Zhao, J.; Yu, D.; Shafran, I.; Narasimhan, K. R.; and Cao, Y. 2022. React: Synergizing reasoning and acting in language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Zhang, D.; Lin, Y.; Wu, Z.; Sun, Y.; Li, B.; Li, D.; and Peng, H. 2026. Useful Memories Become Faulty When Continuously Updated by LLMs. *arXiv preprint arXiv:2605.12978*.

Zhang, Q.; Hu, C.; Upasani, S.; Ma, B.; Hong, F.; Kamanuru, V.; Rainton, J.; Wu, C.; Ji, M.; Li, H.; Thakker, U.; Zou, J.; and Olukotun, K. 2025. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models. *arXiv preprint arXiv:2510.04618*.

Zhao, A.; Huang, D.; Xu, Q.; Lin, M.; Liu, Y.-J.; and Huang, G. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 19632–19642.

This appendix accompanies the paper *CONTRAMEM: Learning Self-Evolving Procedural Memory from Contrasting Multi-Model Trajectories*. Appendix A provides additional experimental detail and the formal pipeline; Appendix B provides memory schemas, rendered card examples, and the complete construction and runtime prompts for both benchmarks. Table and figure numbers match the references used in the main paper.

Additional Experimental Detail

A.1 Memory-Bank Evolution and Frozen Configuration

Figure A1 summarizes how the two memory banks evolve under localized curation, and Table A1 records the frozen configuration used for all main-table runs.

Time uses a benchmark-specific temporal adapter because asynchronous obligations are evaluated against a simulated scenario clock rather than the wrapper clock. The adapter exposes authoritative simulated time and emits bounded, task-blind tick notifications so that pending obligations can resume. For this ability, pre-tool guidance is disabled; up to two ability-universal Skill Cards may be fixed at task start, and one finalization card may be repeated near the final timer boundary. The tick mechanism itself does not inspect task content or prescribe an application action. The adapter also repairs a wrapper artifact in which a run would terminate on an empty final boundary while obligations remain pending; this repair, like the tick mechanism, is part of the shared Time image and applies identically to both conditions. Task-start Function Cards for Time additionally pass a verb-intent relevance filter, identical across the memory conditions.

A.2 Trajectory Efficiency

Table A2 and Figure A2 report per-ability agent-event statistics for the held-out evaluation summarized in the main text. For Time, both conditions use the normalized temporal runtime; neutral timer ticks make these traces longer in absolute terms. Beyond the aggregate reduction, memory also repairs stalled behavior on Time: hung runs fall from 1 to 0 for Claude Sonnet and from 13 to 6 for DeepSeek V4 Pro, although many resumed executions still fail at later steps absent from the source evidence.

Ability	Avg No	Avg Ours	Δ
Exec	30.3	31.1	+0.8
Search	55.6	40.2	**-15.5**
Ambig	29.5	24.3	**-5.2**
Adapt	19.0	20.7	+1.6
Time	70.9	61.0	**-9.9**
Macro	**41.1**	**35.5**	**-5.6**

Table A2: Agent-event efficiency on GAIA2/ARE held-out tasks, paired by scenario. Negative Δ means that ContraMem executes the task with fewer agent-level events.

A.3 Reference-Split Results

Table A3 reports the same comparison as the main held-out table, but on the reference split used to build memory. These results are not the primary generalization metric; they verify that the memory bank improves the tasks from which source trajectories were collected without being evaluated by same-scenario trajectory replay.

The reference split shows the same qualitative pattern as the held-out split: ContraMem raises aggregate success from 22.8% to 44.8%, with the largest gains on procedural tasks requiring bounded candidate sets, complete target coverage, or safe branch separation. Time remains weak even here, again indicating that static procedural memory does not replace a runtime controller for asynchronous obligations.

A.4 Curator Ablation

Table A4 reports the full Curator ablation. The append-only baseline commits every raw Skill Card candidate emitted by the Reflector as Add, while ContraMem applies Curator edits before runtime retrieval. Curation reduces the four-bank Skill Card count from 273 to 205 while increasing GPT-5.5 held-out success from 104/160 to 115/160: the Curator's advantage does not come from added memory volume: it removes redundant or overly broad cards and preserves sharper runtime guidance.

A.5 Prior Memory Baseline Results

Table A5 reports the complete numerical comparison with prior memory systems on GPT-5.5—the no-memory anchor, per-ability values, and macro averages—underlying the figure shown in the main paper's ablation section.

A.6 ACE Matched-Evidence Reproduction

Fidelity target. We adapt the released ACE AppWorld pipeline (Zhang et al. 2025), pinned to ACE commit `bcb7cea` and AppWorld submodule `9f3e921`. We preserve its learning unit: the Reflector sees one trajectory and the complete current playbook; the Curator emits Add or no-op. Following the pinned default and no-ground-truth configuration, construction uses one epoch and the released add-only path, not paper-only grow-and-refine operations. We add no contrast packets, card types, semantic merges, retrieval, or pre-tool/JIT guidance. We therefore report an evidence-matched GAIA2 control, not a reproduction of ACE's original AppWorld score.

GAIA2 interface adaptation. Only benchmark interfaces change: AppWorld code cells map to ordered GAIA2 function calls, REPL outputs and exceptions to observations and errors, and `Supervisor.complete_task` to the final response. The ability-specific playbook is written in full to the task-start agent context, with no retrieval or dynamic reinjection. Construction prompts exclude source identities, scenario identifiers, raw judge prose, oracle actions, hidden answers, and expected writes. We retain ACE's full-playbook policy rather than token-matching it to ContraMem's top-k renderer.

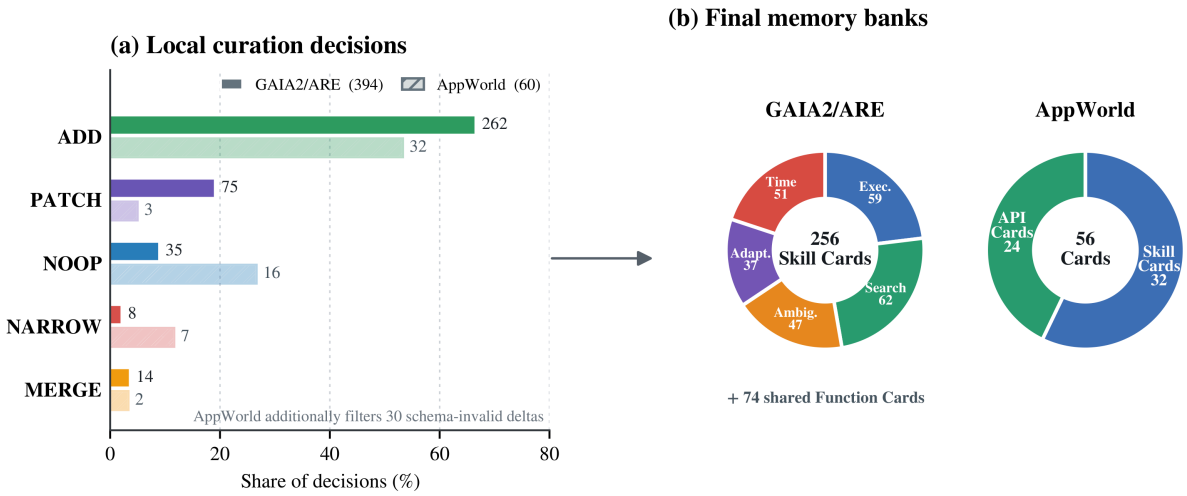

Figure A1: Memory-bank evolution across benchmarks. (a) Distribution of localized Skill Curator decisions; numbers at bar ends are counts. Solid bars denote GAIA2/ARE (394 decisions) and hatched bars AppWorld (60). AppWorld additionally filters 30 schema-invalid candidate deltas before curation. (b) Final banks: GAIA2/ARE contains 256 ability-specific Skill Cards plus 74 shared Function Cards; AppWorld contains 32 Skill Cards and 24 Function/API Cards.

Item	Configuration
Dataset	GAIA2/ARE dynamic app-agent tasks.
Split	Balanced seed 20260610; 40 reference and 40 held-out tasks per ability.
Abilities	Execution, Search, Ambiguity, Adaptability, and Time.
Memory source agents	GPT-5.5, Claude Sonnet 4.6, and DeepSeek V4 Pro.
Target agents	GPT-5.5, Claude Sonnet 4.6, DeepSeek V4 Pro, and Qwen3.7 Plus.
Memory construction model	GPT-5.5 with high reasoning effort for Function Card construction, Skill reflection, and Skill curation.
Final memory bank	256 Skill Cards (including the 51-card Time bank) and 74 global Function Cards.
GPT-5.5 target setting	Default evaluation setting; no explicit extended-thinking override at runtime.
Claude Sonnet target setting	High/hard extended-thinking setting for Claude Sonnet 4.6 target runs.
Verifier	GAIA2 verifier with GPT-5.5 judge where LLM judgment is required.
Runtime	Standard GAIA2/ARE integration for the four non-temporal abilities; Time uses the temporal runtime described below.
Skill retrieval	BM25 over Skill Card title, trigger, tags, and core rule, with a soft app-family prior.
Skill injection	Up to three task-start Skill Cards; Time may reserve up to two slots for ability-universal cards.
Function injection	Retrieved Function Cards plus pre-tool function guidance; pre-tool guidance is disabled for Time.
Memory policy	Retrieved memories are soft guidance; live environment observations remain ground truth.
Timeout	300s per-scenario runner timeout; 900s for Time under the revised temporal runtime (both conditions).
Concurrency	Two parallel scenarios unless otherwise stated in a run manifest.

Table A1: Frozen evaluation configuration used for the main results.

Matched evidence and evaluation. We hold fixed the trajectory multiset, construction and target models, compact outcome feedback, held-out scenarios, and evaluation harness. ACE receives the same 120 reference trajectories per ability (40 tasks times three source models), but processes each independently and never observes same-task attempts together. GPT-5.5 with high reasoning effort builds each playbook; the frozen GPT-5.5 target is evaluated once on the same 40 held-out scenarios. The resulting Execution, Search, and Ambiguity playbooks contain 48, 54, and 56 bullets (approximately 3,762, 3,904, and 4,212 tokens), respectively; held-out success appears in the main paper's baseline figure.

A.7 AWM Offline Reproduction and Baseline Accounting

Fidelity target. We adapt the released AWM code (Wang et al. 2024), pinned to commit `8c0ff8c`. Prompt semantics and workflow syntax follow the released WebArena instruction, while the single grouped induction call and frozen test-time library follow AWM's offline setting and released Mind2Web path. Each demonstration contains a task and ordered successful action trajectory; the builder abstracts recurring sub-routines, replaces instance values with variables, and emits workflows containing at least two actions. The complete ability library is injected once at task start. We add no failure reflection, same-task contrast, Function/Skill

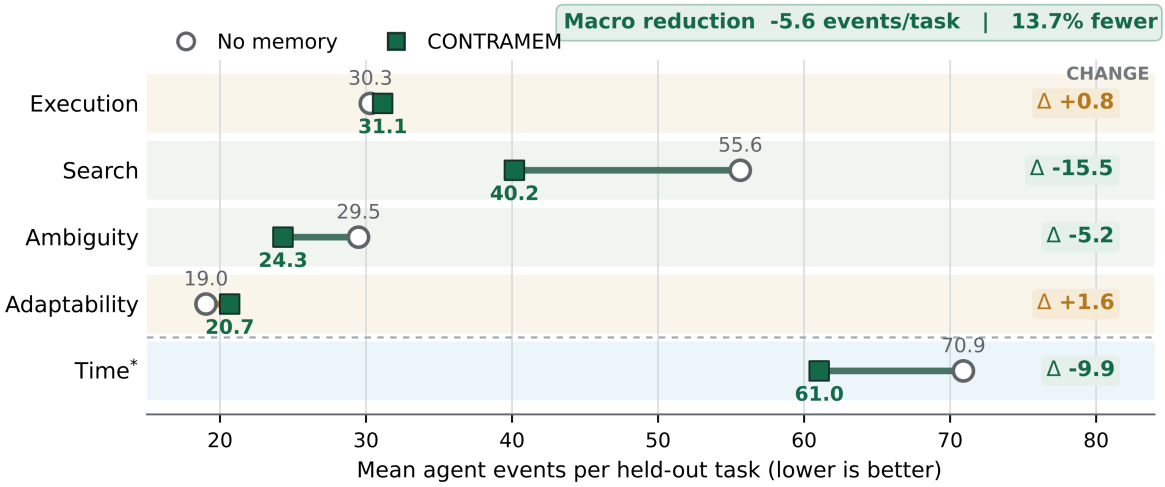

Figure A2: Mean agent events per held-out task, paired by scenario. Lower is better. Small increases on Execution and Adaptability reflect additional necessary work completed by successful memory-conditioned agents.

Target	Method	Exec.	Search	Ambig.	Adapt.	Time	Overall
GPT-5.5	No memory	45.0	52.5	10.0	17.5	2.5	25.5
	ContraMem	**70.0** (+25.0)	**92.5** (+40.0)	**60.0** (+50.0)	**35.0** (+17.5)	**10.0** (+7.5)	**53.5** (+28.0)
Claude Sonnet	No memory	37.5	57.5	20.0	20.0	2.5	27.5
	ContraMem	**70.0** (+32.5)	**70.0** (+12.5)	**52.5** (+32.5)	**50.0** (+30.0)	0.0 (-2.5)	**48.5** (+21.0)
DeepSeek V4 Pro	No memory	32.5	47.5	12.5	17.5	0.0	22.0
	ContraMem	**72.5** (+40.0)	**72.5** (+25.0)	**42.5** (+30.0)	**47.5** (+30.0)	0.0 (+0.0)	**47.0** (+25.0)
Qwen3.7 Plus	No memory	12.5	42.5	7.5	17.5	0.0	16.0
	ContraMem	**37.5** (+25.0)	**60.0** (+17.5)	**20.0** (+12.5)	**32.5** (+15.0)	0.0 (+0.0)	**30.0** (+14.0)
	Aggregate	31.9 → **62.5**	50.0 → **73.8**	12.5 → **43.8**	18.1 → **41.3**	1.3 → **2.5**	22.8 → **44.8**

Table A3: Reference-split success rates on GAIA2/ARE. Each ability contains 40 reference tasks per target model. Numbers are percentages; parentheses show absolute improvement over the no-memory baseline.

Cards, Curator, retrieval, pre-tool hook, JIT delivery, or online update.

Why offline AWM. Online AWM updates memory while traversing the evaluation stream and therefore depends on earlier held-out tasks, their order, and an auxiliary success evaluator. That transductive protocol is not comparable to the frozen reference-build/held-out-evaluate pass@1 cells used throughout this paper. Moreover, GAIA2's in-container verifier result is not exposed to the acting agent as a test-time learning signal. We therefore evaluate the official offline mechanism and do not present AWM's WebArena online headline as the matched baseline.

GAIA2 adaptation and evidence. Website groups map to abilities and browser actions to ordered GAIA2 API calls. We considered app-level grouping, but GAIA2 tasks routinely join several apps, so it would either duplicate a trajectory across libraries or discard cross-app ordering. AWM starts from the same 120-trajectory source pool per ability as ContraMem, then applies its native success-only rule. This leaves 46 Execution, 63 Search, and 17 Ambiguity demonstrations spanning 19, 31, and 10 distinct reference scenarios. GPT-5.5 with high reasoning effort performs one induction call per ability, producing 21, 19, and 15 workflows (approximately 3,334, 3,100, and 2,620 rendered tokens). The Search induction call emitted 20 candidate workflows; deterministic format validation retained 19 after excluding one single-action block that did not satisfy the released minimum-step criterion. Thus, AWM has the same *source pool*, not the same number of usable demonstrations. Before induction, exact names, identifiers, final answers, and verifier text are masked while preserving API names, action order, and generic result shapes; source-model identity is never rendered.

Evaluation and outcome. The frozen GPT-5.5 target runs once on the same 40 held-out scenarios per ability, with the complete ability-level workflow library supplied at task start and no retrieval or dynamic injection. AWM attains 42.5%, 85.0%, and 15.0% on Execution, Search, and Ambiguity (47.5% macro). Its paired improvement over no memory is significant in aggregate (17 vs. 5 discordant flips, p=0.0169), but ContraMem remains 30.0 points higher, with 38 vs. 2 discordant flips (p=1.49 × 10^{-9}).

Ability	Raw Cards	Curated Cards	Card Reduction	Append-Only Success	Curated Success
Search	74	62	16.2%	39/40 (97.5%)	**40/40 (100.0%)**
Ambiguity	69	47	31.9%	19/40 (47.5%)	**21/40 (52.5%)**
Execution	75	59	21.3%	26/40 (65.0%)	**32/40 (80.0%)**
Adaptability	55	37	32.7%	20/40 (50.0%)	**22/40 (55.0%)**
Total	**273**	**205**	**24.9%**	**104/160 (65.0%)**	**115/160 (71.9%)**

Table A4: Curator ablation on GPT-5.5 held-out tasks. Append-only commits every Reflector delta as ADD, skipping local consolidation; CONTRAMEM uses the Curator to merge duplicates, narrow triggers, reject weak deltas, and keep compact transferable cards. The ablation covers the four primary non-temporal abilities for which append-only banks were constructed.

Method	Exec.	Search	Ambig.	Macro
No memory	47.5	52.5	12.5	37.5
Raw retrieval	45.0	75.0	17.5	45.8
AWM	42.5	85.0	15.0	47.5
ACE	50.0	70.0	12.5	44.2
CONTRAMEM	**80.0**	**100.0**	**52.5**	**77.5**

Table A5: Exact GPT-5.5 held-out success rates underlying the main paper's baseline figure. Macro is the unweighted mean over Execution, Search, and Ambiguity. Each cell contains the same 40 held-out scenarios.

Runtime and construction accounting. Table A7 compares AWM, ACE, and CONTRAMEM using actual target-agent traces. No token-budget matching is imposed: AWM and ACE expose their complete ability memories, whereas CONTRAMEM retrieves a compact task-conditioned card set. AWM is inexpensive to build and improves Search strongly, but its full-library runtime still uses 22.93M target tokens across 120 tasks, compared with 20.56M for CONTRAMEM. The corresponding averages are 35.2 versus 31.3 agent events per task. Agent events count benchmark-level reads, writes, observations, and agent actions; they are distinct from LLM calls. Verifier usage is omitted because judge tokens are not recorded in the agent traces. The construction panel shows that AWM uses one grouped call and 0.142M tokens ($0.92), ACE uses 246 incremental calls and 15.72M tokens ($94.57), and CONTRAMEM uses approximately 1.44M tokens for the Search Skill bank; even charging the entire shared five-ability Function bank to Search yields approximately 1.94M tokens. The comparison therefore separates construction economy from downstream quality: AWM is cheapest, while CONTRAMEM achieves the highest held-out success.

A.8 Pipeline Algorithms and Retrieval Scoring

Algorithms 1 and 2 give the formal pipeline behind the *Methodology* section of the main paper. Both benchmarks share the same construction and retrieval logic; Table A6 summarizes their interface-specific instantiations.

Skill retrieval scores lexical relevance with a soft app-family prior,

$$s(c;x) = \mathrm{BM25}\big(q(x),\, d(c)\big) + \lambda\, \mathbf{1}\big[\mathrm{apps}(c) \cap \mathrm{apps}(x) \neq \emptyset\big], \tag{1}$$

Algorithm 1 Offline contrastive bank construction

Require: reference sets $\{\mathcal{D}^a_{\mathrm{ref}}\}$ and source agents $\mathcal{M}$
Require: schema, privacy, and evidence validator Π_V
$\mathcal{F} \leftarrow \emptyset$; $\mathcal{S}^a \leftarrow \emptyset$ for every ability a
for all abilities a and tasks $x_i \in \mathcal{D}^a_{\mathrm{ref}}$ **do**
  $T_i \leftarrow \{\mathrm{Normalize}(\tau_{i,m}) : m \in \mathcal{M}\}$
  $\mathcal{O} \leftarrow \mathcal{O} \cup \mathrm{FuncObs}(T_i)$
  $P_i \leftarrow \mathrm{Packet}(T_i)$
end for
for all observed functions u **do**
  $f_u \leftarrow \textsc{Builder}(\mathcal{O}_u)$
  if $\Pi_V(f_u)$ **then**
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{f_u\}$
  end if
end for
for all packets P_i in curriculum order **do**
  $R_i \leftarrow \mathrm{retrieve}(\mathcal{S}^a, P_i)$
  $\Delta \leftarrow \textsc{Reflector}(P_i, R_i, \phi^a)$, $|\Delta| \leq 3$
  for all $\delta \in \Delta$ with $\Pi_V(\delta)$ **do**
    $R_\delta \leftarrow \mathrm{retrieve}(\mathcal{S}^a, \delta)$
    $op \leftarrow \textsc{Curator}(\delta, R_\delta)$
    $op \in \{\textsc{Add}, \textsc{Patch}, \textsc{Merge}, \textsc{Narrow}, \textsc{Noop}\}$
    $\mathcal{S}^a \leftarrow \Pi_V\big(\mathcal{S}^a \oplus op(\delta)\big)$
  end for
end for
return $\mathcal{B} = (\mathcal{F}, \{\mathcal{S}^a\})$

where $q(x)$ is the task query, $d(c)$ indexes the card's title, trigger, tags, and core rule, and the prior weight λ boosts but never gates retrieval.

Guarded updates. The Curator applies the selected operation through $\oplus$ (symbols in Algorithms 1–2): ADD inserts a card; PATCH, MERGE, and NARROW replace their target cards; and NOOP leaves the bank unchanged. Projection through Π_V rejects invalid cards and patches unsupported by the delta, the targeted cards, or related Function Cards.

	GAIA2/ARE	AppWorld
Focus selector	Ability label	Deterministic task signature
Skill bank	Ability-specific banks	Global, signature-tagged bank
Function memory	App-tool contracts	API doc-deltas and misuse guards
Runtime delivery	Task-start skills; pre-tool Function Cards	Task-start cards (C1a)

Table A6: Benchmark-specific instantiations of the shared construction and retrieval pipeline.

(a) Held-out runtime

Ability	Method	Success (%)	Memory tok.	Target tok./task (k)	Calls /task	Events /task	Cost (USD)
Execution	AWM	42.5	3,334	223.5	10.7	30.5	18.98
	ACE	50.0	3,862	267.0	12.6	43.4	23.85
	ContraMem	**80.0**	1,879	218.5	10.5	30.6	26.71
Search	AWM	85.0	3,100	160.4	9.5	49.5	16.20
	ACE	70.0	4,005	145.6	9.8	93.4	17.15
	ContraMem	**100.0**	1,836	126.2	8.9	39.1	16.08
Ambiguity	AWM	15.0	2,620	189.3	9.7	25.6	19.92
	ACE	12.5	3,769	195.9	9.9	26.2	21.21
	ContraMem	**52.5**	1,837	169.3	10.2	24.2	14.84
Aggregate	AWM	47.5	3,018	191.1	10.0	35.2	55.10
	ACE	44.2	3,879	202.8	10.7	54.3	62.21
	ContraMem	**77.5**	1,851	171.4	9.8	31.3	57.64

(b) Offline Search construction

Build component	Calls	Tokens	Cost (USD)
AWM Search workflow bank	1	0.142M	0.92
ACE Search playbook	246	15.72M	94.57
ContraMem Search Skill bank	114	∼1.44M	∼14.09
Shared Function bank (all abilities)	74	∼0.50M	∼3.17
ContraMem conservative charge	188	∼1.94M	∼17.26

Table A7: Runtime and construction footprint of structured experience-memory baselines. (a) GPT-5.5 held-out runtime: target tokens and calls are provider-reported agent usage, while agent events are benchmark-level trajectory events. Cost covers all 40 tasks per ability (120 for Aggregate). (b) Search construction: all methods start from the same 120-trajectory source pool; AWM's native success-only filter retains 63 demonstrations. Tildes denote reconstructed usage estimates. Efficiency columns are interpreted jointly with success because shorter failing runs can be artificially cheap.

Algorithm 2 Runtime retrieval and injection with a frozen bank

Require: task x, bank $\mathcal{B} = (\mathcal{F}, \{\mathcal{S}^a\})$
Require: retrieval caps k_s, k_f and runtime profile
$a \leftarrow$ ability label of x (or all banks if unlabeled)
$C_s \leftarrow$ top-$k_s\{c \in \mathcal{S}^a : s(c; x)\}$ (Eq. 1)
$C_f \leftarrow$ top-k_f relevant cards from $\mathcal{F}$
inject $\rho(C_s \cup C_f)$ once at task start as soft guidance
if the runtime profile enables pre-tool delivery **then**
  while the agent proposes an app call u **do**
    if $f_u \in \mathcal{F}$ **then**
      surface f_u immediately before the call
    end if
  end while
end if

Symbols.

$\mathcal{F}$	Global Function Card bank
$\mathcal{S}^a$	Skill Card bank for ability a
P_i	Same-task contrast packet
Δ	Candidate deltas from the Reflector (≤ 3)
ρ	Compact runtime renderer
Π_V	Schema, privacy, and evidence validator

Reproducibility Plates

This appendix consolidates the shared memory schemas, rendered card examples, and the benchmark-specific prompts used for construction and runtime injection. Sections B.3–B.6 report the GAIA2/ARE suite. Sections B.7–B.11 reproduce the four core AppWorld prompts and summarize the focus branch selected within the Reflector. The AppWorld plates reflect the frozen, task-start-only condition used in the reported evaluation; online updates and the optional just-in-time delivery variant are intentionally excluded. AppWorld exposes a stateful Python REPL, live API documentation, exceptions, and `apis.supervisor.complete_task`. All plates are monochrome so that field boundaries and invariants remain legible in print.

APPENDIX B.1 | Memory Schemas

The three memory schemas are field-level contracts. Function Cards remain tool-local, Skill Deltas carry procedural contrast, and Curator Patches expose the smallest permitted bank edit; provenance and construction reasoning stay outside runtime memory.

Function Card schema

`card_id`	`function::App.function`
`tool`	Exact `App.function` identity
`app`	Application family
`function`	Callable function name
`what_it_does`	One observed capability statement
`arguments`	Observed names, meanings, and value forms
`returns`	Return kind and useful fields
`usage_rules`	Function-local calling guidance
`common_mistakes`	Evidence-supported misuse guards
`side_effects`	Observed state change, if any

APPENDIX B.1 | Memory Schemas (continued)

Skill Delta schema

`title`	Short transferable name
`applies_when`	Observable retrieval trigger
`solves`	Reusable problem addressed
`tags`	Compact retrieval terms
`skill`	Core rule, completion condition, success/failure contrast, recovery, efficiency
`evidence`	Feedback, transition, compact good/bad examples
`functions_used`	Relevant `App.function` names

Curator Patch schema

`delta_index`	Candidate being edited
`operation`	`ADD`, `PATCH`, `MERGE`, `NARROW`, or `NOOP`
`target_card_ids`	Existing cards affected
`retained_card_id`	Card preserved after an edit
`new_or_updated_card`	Local replacement card or `null`
`reason`	Concise evidence-grounded justification

APPENDIX B.2 | Rendered Card Examples

Function Card · tool-level contract

TOOL

`EmailClientV2.delete_email`

WHAT IT DOES

Deletes one specified email from one specified folder; observed successful use removes emails from `INBOX` by exact `email_id`.

ARGUMENTS

- `email_id`: exact identifier from a prior listing or search result.
- `folder_name`: folder containing the target email; observed value `INBOX`.

Side effect (destructive)

Removes the targeted email. Because this is a destructive write, filtering and selection must be verified before the call.

USAGE RULES

- Use exact `email_id` values from observations.
- Call once per `email_id` for multi-delete tasks.
- Pass the folder where the email was found.

Mistake guard

✗ Do not call `delete_email` when the intended action is moving, keeping, or only clarifying about an email.

Skill Card · task-level procedure

TITLE

Delete clear emails despite an ambiguous keep exception.

APPLIES WHEN

A destructive email task has fully specified base search criteria, but a keep/exception clause matches multiple possible records.

Core decision boundary

Partition before writing: delete clear non-exceptions, preserve every possible exception candidate, and ask only for the unresolved exception decision.

COMPLETION LEDGER

- ✓ Base matches are split into clear-delete and ambiguous-exception records.
- ✓ Each clear-delete record is deleted exactly once.
- ✓ No ambiguous exception candidate is deleted.

CONTRASTIVE EVIDENCE

Observed passing pattern

✓ Passing trajectories delete safe non-exception candidates, then ask which exception was intended.

Observed failing pattern

✗ Failing trajectories detect ambiguity, ask immediately, and omit all safe delete writes.

FUNCTIONS USED

`search_emails`, `delete_email`, `send_message`

Figure B1: Runtime renderings of one Function Card (blue) and one Skill Card (green) exactly as the target agent receives them. The Function Card records a callable tool contract with destructive-write guards; the Skill Card records the decision boundary, completion ledger, and contrastive evidence distilled from same-task trajectory contrast. Field schemas appear in Appendix B.1.

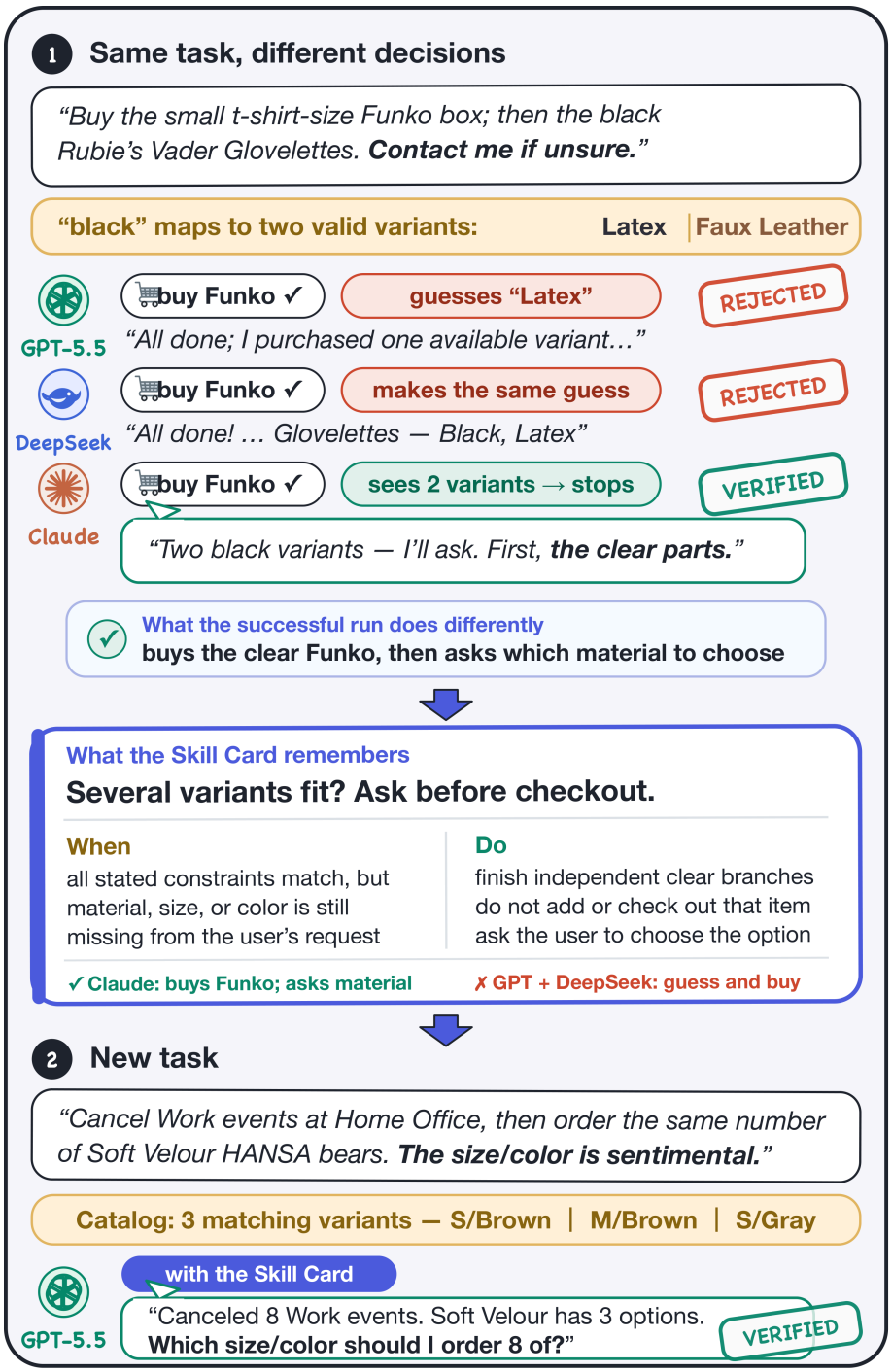

Figure B2: From disagreement to transfer. GPT-5.5 and DeepSeek guess an underspecified shopping variant and fail, whereas Claude completes the safe branch and asks for the missing attribute. The distilled Skill Card retains this decision boundary and guides GPT-5.5 to complete the unambiguous branch and request clarification on a new task.

Function Card Builder

```
# Role & objective
You are a senior tool-use memory curator for autonomous app agents. From observed
  evidence about a single `App.function`, produce one
compact reusable function card. Agents retrieve this card immediately before calling
  the function, so it must support fast tool-use
decisions: what the function does, how to call it correctly, what it returns, what
  side effects it has, and which function-specific
mistakes to avoid. You are documenting one function, not solving the tasks in which it
  appeared.
# Hard rules
These rules are externally validated; violating them invalidates the output.
1. Evidence-grounded: every claim about arguments, return behavior, errors, or side
  effects must follow from the evidence. Never introduce
   an argument, return field, enum value, side effect, or failure mode the evidence
     does not show.
2. Single-function scope: describe only the target function. Do not create
  task-decomposition labels, mini-task labels, or trajectory
   summaries.
3. No guessing: if evidence is insufficient, omit the claim rather than filling space.
  Do not label arguments as required or optional;
   this runtime card stores only observed argument names, meanings, and value forms.
4. De-identified: the evidence may contain concrete scenario values. You may reason
  with them, but the final card must contain none.
   Replace scenario-specific values with typed placeholders: person -> <PERSON>; email
     -> <EMAIL>; phone -> <PHONE>; id/token -> <ID>;
   path/file -> <PATH>; date/time -> <DATE_OR_TIME>; location/address -> <LOCATION>;
     all other scenario-specific text such as product
   names, event titles, subjects, message bodies, search queries, or ZIP codes ->
     <TEXT>. Preserve semantic enum values that are part of
   function behavior, such as INBOX, SENT, TRASH, DRAFT, Work, personal,
     saved_only=true/false.
5. Runtime-only output: return only the compact JSON card. Do not include
  validation_self_check, evidence_refs, trajectory IDs, scenario
   IDs, model names, step IDs, source_event_index, raw judge feedback, raw LLM
     reasoning, or prose outside the JSON.
6. Tool identity: card_id, tool, app, and function must exactly match the provided
  target_tool. For target_tool App.function, use card_id
   "function::App.function", app "App", and function "function".
7. Argument identity: each `arguments[]` item must name exactly one observed argument
  key. Never combine several argument names into one
   string such as "min_price, max_price"; create separate items instead.
# Evidence precedence
- Prefer direct event-level observations over aggregate summaries.
- Use aggregate statistics to identify repeated patterns, not to override direct
  evidence.
- If direct observations conflict and the conflict cannot be resolved, omit the claim
  or abstain.
# Writing guidance
- Keep observed contract and inferred advice separate: arguments, returns, and
  side_effects describe evidence; usage_rules and
  common_mistakes describe how future agents should act.
- Include a common_mistake only when the failure is attributable to this exact
```

```
  function call, not upstream reasoning, final-answer
  wording, date interpretation, missing filtering, or unrelated tools.
- Be compact but complete. Preserve useful result shape, enum values, pagination
  behavior, and real function-specific errors. Avoid
  padding and repetition.
- Prefer concrete tool-use guidance over vague caution. Good: "Use offset/limit
  pagination when result metadata indicates more records are
  available." Bad: "Be careful when using this function."
Return exactly one valid JSON object. No markdown, no code fences.
Required compact runtime schema:
{
  "card_id": "function::App.function",
  "tool": "App.function",
  "app": "App",
  "function": "function",
  "what_it_does": "One concise evidence-grounded sentence or short paragraph.",
  "arguments": [
    {
      "name": "...",
      "meaning": "...",
      "value_form": "<PLACEHOLDER_OR_ENUM_DESCRIPTION>"
    }
  ],
  "returns": "Short description of observed return kind and key useful fields or empty
    behavior.",
  "usage_rules": [
    "Concrete function-specific guidance."
  ],
  "common_mistakes": [
    "Function-specific mistake to avoid, if supported."
  ],
  "side_effects": [
    "Observed write or external side effect, if any."
  ]
}
Field limits:
- arguments: at most 12 items.
- usage_rules: at most 5 short bullets.
- common_mistakes: at most 4 short bullets.
- side_effects: at most 4 short bullets.
- If no common mistake or side effect is supported, return [] for that field.
- Keep "returns" as a string, not a nested object.
INPUT_START
target_tool:
{{target_tool}}
deterministic_aggregate_stats:
{{deterministic_aggregate_stats_json}}
final_function_evidence_package:
{{final_function_evidence_package_json}}
INPUT_END
Now produce the compact JSON card.
```

APPENDIX B.4 | GAIA2/ARE **1/2**

Cross-Trajectory Skill Reflector

```
# Role and Objective
You are a Cross-Trajectory Skill Reflector for a GAIA2 / ARE app agent.
You receive one `task_contrast_packet`: the same task attempted by several models,
  with observed tool spans, outcomes, verifier or judge
feedback, passed/failed final messages, write summaries, and cross-trajectory
  differences.
Your main job is contrastive procedural memory distillation: extract the smallest
  reusable skill insights that explain why stronger runs
succeeded, why weaker runs failed, how to recover once the failure state appears, or
  how a shorter successful run avoided unnecessary
work.
Return at most 3 candidate skill deltas. Prefer one precise, evidence-grounded delta
  over several broad warnings.
The schema below is your complete output contract: add nothing outside it.
# Reflection Protocol
Reason through each candidate in this order before writing JSON:
1. Locate the decision boundary. Find the observable point where trajectories
  diverged: target selection, candidate-set construction,
   join/filter order, state change, write action, clarification, wait/deadline, or
     final response.
2. Attribute the outcome. Explain why that divergence drove success, failure,
  recovery, or efficiency. Isolate the causal decision; do not
   recap the whole trajectory.
3. Abstract exactly one level. The insight should be more general than this scenario,
  but still concrete enough to guide a future agent.
   Avoid slogans such as "be careful" or "verify everything" unless the trigger and
     corrective action are explicit.
4. State the core rule and completion condition. `core_rule` is one concise if-then,
  ordering, obligation, recovery, or stopping rule the
   future agent should follow. `completion_condition` states when the task, branch,
     wait, search, or write obligation is satisfied enough
   to stop or move on. This is especially important for Search, Time, and Execution
     tasks.
5. Encode only supported contrast. Each insight may fill only the contrast fields
  supported by the packet:
   - `success_contrast`: why stronger runs won; capture decisions, orderings, checks,
     or response shapes worth repeating.
   - `failure_contrast`: reusable root-cause mistakes in weaker runs. Do not mirror
     `success_contrast`; state a splitting decision once,
     on the more informative side.
   - `recovery`: what to do once the failure state appears, including self-recovery
     observed in a trajectory or advice justified by the
     contrast.
   - `efficiency`: what shorter successful runs skipped, batched, bounded, or stopped
     doing without losing correctness. Use only when
     successful runs differ meaningfully in length or redundancy.
6. Separate lesson, evidence, retrieval keys, and tools. `skill` stores the
  transferable lesson. `evidence` stores observed feedback,
   state transitions, and good/bad response shapes.
   `tags` are 3-8 compact lowercase retrieval keywords chosen from your understanding
     of the insight. They help Python retrieve this
   memory later with keyword overlap. Avoid private values and exact task answers
     unless they are semantically essential to the lesson.
   `functions_used` stores concrete `App.function` names observed in the packet or
     current memory that are relevant to the insight. Do not
```

Cross-Trajectory Skill Reflector

```
   invent functions. These are observed tools, not role labels and not a function
     manual.
7. Respect all-failed evidence. If all trajectories failed, produce diagnostic,
  failure, or recovery deltas only. Never invent a success
   recipe.
8. Keep the output small. Return at most 3 deltas. Prefer one sharp transferable
  insight over several generic warnings.
# Ability Reflection Focus
Each GAIA2 capability pairs an intended behavior with the failure modes to mine for:
{{reflection_focus_block}}
# Grounding and Privacy
- Ground every lesson in verifier feedback, passed/failed messages, oracle-style
  examples, tool spans, write summaries, or state changes.
  Unsupported lessons should be omitted.
- Do not document function arguments, return schemas, or low-level syntax; that
  belongs to function cards.
- Do not output evidence refs, trajectory IDs, scenario IDs, step IDs, model names,
  raw judge prose, or copied private text.
- Generalize concrete values: `<PERSON>`, `<EMAIL>`, `<PHONE>`, `<ID>`,
  `<DATE_OR_TIME>`, `<PRODUCT_TEXT>`, `<TITLE_TEXT>`, `<TEXT>`.
- De-identified examples are allowed only when they teach a write, clarification, or
  final-response shape.
# Input
task_contrast_packet:
{{task_contrast_packet_json}}
current_relevant_memory:
{{relevant_skill_cards_json}}
# Output
Return exactly one valid JSON object: no markdown, no code fences.
{
  "deltas": [
    {
      "title": "short transferable title",
      "applies_when": "trigger state",
      "solves": "reusable problem",
      "tags": ["keyword"],
      "skill": {
        "core_rule": "if-then, ordering, obligation, recovery, or stopping rule",
        "completion_condition": ["done condition"],
        "success_contrast": ["success pattern"],
        "failure_contrast": ["failure pattern"],
        "recovery": ["recovery action"],
        "efficiency": ["efficiency lesson"]
      },
      "evidence": {
        "observed_feedback": ["feedback label"],
        "state_transition": ["before -> after"],
        "good_examples": ["good response shape"],
        "bad_examples": ["bad response shape"]
      },
      "functions_used": ["App.function"]
    }
  ]
}
```

Skill-Memory Curator

```
You are a Skill-Memory Curator for a GAIA2 / ARE app agent.
You receive:
- current relevant skill cards retrieved from the memory bank
- new candidate skill deltas proposed by the Reflector
- compact function cards for observed `App.function` names
Your job is memory editing, not new reflection. Compare each new delta against the
  current relevant memory and decide whether it should be
added, merged, patched, narrowed, rejected, or ignored. Keep edits local. Do not
  rewrite the whole memory bank.
# Curator Protocol
Process every input delta exactly once.
1. Compare the delta with the retrieved cards and choose the smallest valid operation.
2. Preserve semantic closure. You may only combine, compress, narrow, or rephrase
  content supported by:
   - the candidate delta,
   - the targeted current cards,
   - the supplied function cards.
   Do not introduce new rules, failure modes, recovery actions, completion conditions,
     examples, or functions.
3. Preserve scope.
   - Function names may appear only in `functions_used`.
   - Do not include arguments, return schemas, or side effects.
   - Do not broaden a card merely to absorb a delta.
   - If only the trigger is too broad, use `NARROW_TRIGGER`.
4. Preserve privacy and compactness. De-identify source-specific values and produce a
  concise resulting card.
# Operation and Patch Rules
- `ADD_CARD`: use when no semantically equivalent card exists. Provide the complete
  new card.
- `PATCH_CARD`: use when the delta adds supported content without materially changing
  card scope. Provide the complete updated card.
- `MERGE_CARD`: use when two or more existing cards express substantially the same
  transferable rule. Include all involved cards in
  `target_card_ids`, set `target_card_id` to the retained canonical card, and provide
    the complete merged card.
- `NARROW_TRIGGER`: use when the existing rule is useful but `applies_when` is too
  broad. Provide the complete narrowed card.
- `NOOP`: use when the delta adds no meaningful new information, evidence is too weak
  or conflicting, or the delta is only function
  syntax. `new_or_updated_card` must be null.
Do not silently broaden a card's scope. If the new evidence applies only under a
  narrower condition, use `NARROW_TRIGGER` rather than
```

```
`PATCH_CARD`.
# Input
current_relevant_skill_cards:
{{current_relevant_skill_cards_json}}
new_deltas:
{{skill_deltas_json}}
related_function_cards:
{{related_function_cards_json}}
# Output
Return exactly one valid JSON object. No markdown. No code fences.
{
  "patches": [
    {
      "operation": "ADD_CARD",
      "target_card_id": "existing skill::skill_<hex> id or null",
      "target_card_ids": ["skill::skill_<hex>"],
      "new_or_updated_card": {
        "title": "short transferable title",
        "applies_when": "trigger state",
        "solves": "reusable problem",
        "tags": ["keyword"],
        "skill": {
          "core_rule": "if-then, ordering, obligation, recovery, or stopping rule",
          "completion_condition": ["done condition"],
          "success_contrast": ["success pattern"],
          "failure_contrast": ["failure pattern"],
          "recovery": ["recovery action"],
          "efficiency": ["efficiency lesson"]
        },
        "evidence": {
          "observed_feedback": ["feedback label"],
          "state_transition": ["before -> after"],
          "good_examples": ["good response shape"],
          "bad_examples": ["bad response shape"]
        },
        "functions_used": ["App.function"]
      },
      "reason": "one concise reason"
    }
  ]
}
```

Runtime Memory Injection

```
TASK-START RUNTIME MEMORY TEMPLATE
## Runtime Skill and Function Memory (RTM)
These memories were selected by deterministic retrieval from previous cross-model
  trajectories. They are soft procedural guidance, not
facts about the current environment. Current tool observations are always ground
  truth.
Task-start skill memory:
1. {{skill_1.runtime_prompt}} Response contract:
  {{skill_1.response_contract_if_present}}
2. {{skill_2.runtime_prompt}} Response contract:
  {{skill_2.response_contract_if_present}}
3. {{skill_3.runtime_prompt}} Response contract:
  {{skill_3.response_contract_if_present}}
Relevant function memory:
- [score={{score}}] Tool Memory for {{App.function}}:
  {{what_it_does}} Use: {{usage_rule}} Avoid: {{common_mistake}}
  Side effect: {{side_effect}}
Memory-use rules:
- Skill memory decides when/why to write or block; function memory only explains how a
  tool is called.
- Apply memory only when its trigger matches live observations; never replay old
  values.
- For ambiguity or multi-write tasks, keep a branch ledger: completed, blocked, or
  still unresolved.
- Before any state-changing or user-facing write, verify target/audience, requested
  action, content intent, branch condition, and
  remaining branches.
- For shopping writes, separate catalog product-name words from explicit variant
  options; do not treat words inside the returned product
  name as separate material/color/size constraints unless the user states them
    separately.
- If a branch is blocked, final wording must name the blocked action and missing
  evidence instead of claiming full completion.
- Response-contract examples show wording shape only. Never copy example entities,
  dates, counts, IDs, prices, names, or scenario facts;
  use only live observations.
- For multi-branch ambiguity tasks, give a compact final update: one short
  completed-branch clause only when independent writes were
  actually completed, then one direct unresolved question.
- If ambiguity is among a small set of concrete candidates, include the discriminating
  candidate names/attributes needed for the user to
  choose. Avoid huge lists, incidental IDs, and long narratives.
- Clarification wording contract: if multiple candidates match, say the count and the
  shared discriminating criterion, include the useful
  candidate labels when small, then ask which one to use.
- Empty-set wording contract: if the requested date/filter has no records and
  downstream writes depend on that set, ask whether the user
  meant a different date/filter instead of presenting the whole task as completed.
- Role-recipient wording contract: if a requested email/message recipient is a role or
  unknown contact, ask for that role's
  name/email/contact information; do not merely say you could not identify them.
- Outbound message contract: preserve the requested semantic payload and audience in
```

```
  natural message wording. For group/team messages,
  include an appropriate greeting such as `Hey everyone` or `Hey team` when the
    oracle-style task asks you to message a group.
- Keep final user-facing messages verifier-friendly: preserve requested action,
  audience, content intent, and branch condition; avoid long
  narrative, incidental IDs, or markdown-heavy reports unless the user requested those
    details.
- When you give the final user-facing response, start the message with
  [[reply_to_current]] followed by the actual
  completion/blocked-branch summary.
- Do not run unbounded repeated reads; after repeated empty/irrelevant results, change
  query strategy or proceed with the verified
  candidate set.
PRE-TOOL FUNCTION-MEMORY TEMPLATE
{
  "rtm_pre_tool_memory": "VERIFY_BEFORE_WRITE",
  "executed": false,
  "blocked_command": "{{proposed_command}}",
  "function": "{{App.function}}",
  "memory_seen": {
    "function_card_id": "function::{{App.function}}",
    "function_card_status": "{{shown_now|already_shown}}",
    "function_card_seen_count": "{{count}}",
    "skill_card_status": "{{status}}",
    "skill_card_id": "{{card_id_or_null}}"
  },
  "why": "This is a state-changing Gaia2 action. No app state was changed.
          Re-run the command only after verifying the checklist against live
          observations.",
  "exit_code_intent": "The wrapper exits non-zero after this advisory so
                       chained shell writes do not continue accidentally.",
  "command_args_seen": "{{live_command_arguments}}",
  "detected_risks": ["{{risk_from_live_task_and_state}}"],
  "new_detected_risks": ["{{new_risk_not_previously_shown}}"],
  "observed_candidate_context": "{{live_shopping_context_or_null}}",
  "observed_calendar_context": "{{live_calendar_context_or_null}}",
  "observed_cab_context": "{{live_cab_context_or_null}}",
  "write_checklist": ["{{function_specific_check}}"],
  "function_memory": {
    "status": "{{shown_now|already_shown}}",
    "card_id": "function::{{App.function}}",
    "what_it_does": "{{what_it_does_if_new}}",
    "usage_rules": ["{{usage_rule_if_new}}"],
    "common_mistakes": ["{{common_mistake_if_new}}"],
    "side_effects": ["{{side_effect_if_new}}"]
  },
  "dynamic_skill_memory": "{{optional_matching_pre_write_skill_or_omitted}}",
  "next_step": "{{if a high-confidence risk applies: do not re-run the same
                command unchanged; otherwise: if the command is still exactly
                correct, call it again. Revise arguments, gather missing
                evidence, or report the blocked branch when a check fails.}}"
}
```

Function/API Card Builder

```
# Role & objective
You are a senior tool-use memory curator for autonomous **AppWorld coding agents**.
AppWorld agents write Python code in a stateful REPL and call app APIs as
`apis.<app>.<api>(...)`. They can always read the official API specification at
runtime via `apis.api_docs.show_api_doc(app_name=..., api_name=...)`.
From observed evidence about a single `app.api`, produce one compact reusable
function card. Agents receive this card at task start or right after a step that
first used (or raised an exception on) this API, so it must support fast coding
decisions: how this API actually behaves, which mistakes real agents made with it,
and the safe usage pattern. You are documenting one API, not solving the tasks in
which it appeared.
**Doc-delta principle (AppWorld-specific).** The official doc is provided to you as
`official_doc_context`. The card must NOT restate what the doc already says
(parameter list, types, response schema). Every card claim must be (a) supported by
observed evidence AND (b) additive over the doc: observed empty-result and
end-of-pagination behavior, value-format quirks, auth/token scope in practice,
constraint violations that actually occurred, side effects, and mistakes agents made
despite the doc. If a claim is fully covered by the doc, drop it. Exception: you may
briefly restate a doc constraint inside `common_mistakes` when the evidence shows
agents violating it (e.g., "requested page_limit above the documented max of 20").
# Hard rules
These rules are externally validated; violating them invalidates the output.
1. Evidence-grounded: every claim about arguments, behavior, errors, or side effects
   must follow from the evidence package. Never introduce an argument, return field,
   enum value, side effect, or failure mode that neither the evidence nor the doc
   shows; never introduce doc-only content except as permitted above.
2. Single-function scope: describe only the target `app.api`. No task strategies, no
   trajectory summaries, no multi-API workflows (those belong to skill cards).
3. No guessing: if evidence is insufficient, omit the claim. Do not label arguments
   required/optional (the doc does that); store only observed argument usage.
4. De-identified: evidence may contain concrete world values. Reason with them, but
   the final card must contain none. Placeholders: person -> `<PERSON>`; email ->
   `<EMAIL>`; phone -> `<PHONE>`; token/id -> `<ID>`; date/time -> `<DATE_OR_TIME>`;
   money -> `<AMOUNT>`; titles/names of songs, playlists, products, files ->
   `<TITLE_TEXT>`; other scenario text -> `<TEXT>`. Preserve semantic enum values that
   are part of API behavior (e.g., folder or status enums) and literal `app.api`
   names.
5. Runtime-only output: return only the compact JSON card. No validation self-checks,
   evidence refs, trajectory/task/scenario ids, model names, step indices, raw
   evaluation text, or prose outside the JSON.
6. Tool identity: `card_id`, `tool`, `app`, `function` must exactly match the
   provided `target_tool`. For `spotify.show_playlist_library` use
   `card_id = "function::spotify.show_playlist_library"`.
7. Argument identity: each `arguments[]` item names exactly one observed argument
   key. List only arguments with observed usage worth noting beyond the doc
   (token sourcing, value-form quirks, constraint hits); an empty list is valid.
8. `code_idiom` rules (AST-validated externally; any violation rejects the card):
   at most 5 lines; exactly one `apis.<app>.<api>` call and it must be this card's
   API -- no other `apis.*` access, no imports; pure-Python control flow only; no
   string or number literals except enum values or constraint bounds already named
   in this card; generic variable names only (`items`, `page`, `token`, `i`,
   `result` style) -- never names taken from any trajectory's code; one usage
   pattern, never a multi-step task fragment or task solution. Set `null` when no
   idiom is warranted -- the card must be fully useful without it (idiom rendering
```

```
   can be disabled at runtime).
# Evidence consistency (externally validated)
If `deterministic_aggregate_stats` shows write-method calls (POST/PUT/PATCH/DELETE
beyond login/logout), `side_effects` MUST state the observed effect -- an empty
`side_effects` for an observed write API invalidates the card.
# Evidence precedence
- Direct step-level observations (executed calls, printed outputs, exception
  messages) outrank aggregate statistics.
- Use aggregates to identify repeated patterns, not to override direct evidence.
- Unresolvable conflicts -> omit the claim.
# Writing guidance
- Keep observed contract and advice separate: `what_it_does`/`arguments`/`returns`/
  `side_effects` describe evidence; `usage_rules`/`common_mistakes` tell future
  agents how to act.
- Include a `common_mistake` only when the failure is attributable to this exact API
  call (wrong argument, missing token, constraint violation, misread return), not to
  upstream reasoning, entity resolution, or final-answer wording.
- Prefer concrete guidance over vague caution. Good: "Loop page_index from 0 and stop
  at the first empty page." Bad: "Be careful with pagination."
- Field limits: arguments <= 8; usage_rules <= 5; common_mistakes <= 4; side_effects
  <= 4;
  doc_delta_notes <= 3; `returns` is a string. Empty lists where unsupported.
Return exactly one valid JSON object. No markdown, no code fences.
Required compact runtime schema:
{
  "card_id": "function::app.api",
  "tool": "app.api",
  "app": "app",
  "function": "api",
  "what_it_does": "One concise evidence-grounded sentence focused on observed
    behavior.",
  "arguments": [
    {"name": "...", "meaning": "...", "value_form":
      "<PLACEHOLDER_OR_ENUM_OR_CONSTRAINT_DESCRIPTION>"}
  ],
  "returns": "Observed return kind and useful fields, empty-result and
    end-of-pagination behavior.",
  "usage_rules": ["Concrete function-specific guidance."],
  "common_mistakes": ["Function-specific mistake observed in trajectories."],
  "side_effects": ["Observed write or cross-entity effect, if any."],
  "doc_delta_notes": ["What this card adds beyond the official doc."],
  "code_idiom": "safe usage pattern, <= 5 lines, or null"
}
INPUT_START
target_tool:
{{target_tool}}
official_doc_context:
{{official_api_doc_json}}
deterministic_aggregate_stats:
{{deterministic_aggregate_stats_json}}
final_function_evidence_package:
{{final_function_evidence_package_json}}
INPUT_END
Now produce the compact JSON card.
```

Cross-Trajectory Skill Reflector

```
# Role and Objective
You are a Cross-Trajectory Skill Reflector for an **AppWorld coding agent**.
AppWorld agents solve tasks by writing Python code cells in a stateful REPL: they
consult live API docs (`apis.api_docs.*`), call app APIs as `apis.<app>.<api>(...)`,
observe printed outputs and exception tracebacks, revise code, and finish with
`apis.supervisor.complete_task(answer=...)` (answer omitted for action-only tasks).
Success is judged by unit tests over the final database state and the returned
answer.
You receive one `task_contrast_packet`: the same task attempted by several agents,
with per-trajectory outcomes, compact span sequences (doc lookups, reads, writes,
exceptions, finalization), doc-lookup discipline, write summaries, failed-requirement
labels, and cross-trajectory differences.
Your job is contrastive procedural memory distillation across **four supervision
axes -- correctness (success vs failure), efficiency, recovery, and shared
failure**: same-task differences between heterogeneous agents are the training
signal, not noise. Extract the smallest reusable skill insights that explain why
stronger runs succeeded, why weaker runs failed, how a run recovered once a
failure state appeared, how a shorter successful run avoided unnecessary work --
or, when every agent failed at the same point, what the shared blocker was.
Return at most 3 candidate skill deltas. Prefer one precise, evidence-grounded delta
over several broad warnings. The schema below is your complete output contract: add
nothing outside it.
# Reflection Protocol
Reason through each candidate in this order before writing JSON:
1. Locate the decision boundary. Find the observable point where trajectories
   diverged. In AppWorld this is typically one of: whether an API doc was consulted
   before first use; how an entity was resolved (relationship/id vs name substring);
   whether a paginated collection was exhausted; join/filter order across apps;
   whether a write's target set was verified before writing; whether a written state
   was read back; how an exception was handled; whether stale variables from earlier
   cells were reused after state changed; when and how `complete_task` was called
   (timing, answer presence, answer form).
2. Attribute the outcome. Explain why that divergence drove success, failure,
   recovery, or efficiency. Isolate the causal decision; do not recap trajectories.
3. Abstract exactly one level. More general than this scenario, still concrete
   enough to change a future agent's next code cell. Avoid slogans ("be careful",
   "verify everything") unless trigger and corrective action are explicit.
4. State the core rule and completion condition. `core_rule` is one concise if-then,
   ordering, obligation, recovery, or stopping rule. `completion_condition` states
   when the search, write set, or task is satisfied enough to stop or move on -- for
   answer tasks include the answer-shape obligation (minimal value, numeric for
   counts); for write tasks include the read-back/coverage obligation before
```

```
   `complete_task`.
5. Encode only supported contrast. Fill only fields the packet supports:
   - `success_contrast`: decisions, orderings, checks worth repeating;
   - `failure_contrast`: reusable root-cause mistakes (state a splitting decision
     once, on the more informative side; do not mirror);
   - `recovery`: what to do once the failure state appears (observed self-recovery,
     e.g., exception -> re-read `show_api_doc` -> corrected retry, or advice justified
     by the contrast);
   - `efficiency`: what shorter successful runs skipped, batched, bounded, or
     stopped doing without losing correctness.
6. Optionally add `procedure_sketch`: up to 5 ordered text steps (never code) when
   the ordering itself is the lesson (e.g., resolve entities -> exhaust collection ->
   filter on exact ids -> aggregate once -> complete_task with minimal value).
7. Separate lesson, evidence, retrieval keys, and tools. `skill` stores the
   transferable lesson; `evidence` stores observed feedback labels, state
   transitions, and de-identified good/bad shapes. `tags` are 3-8 lowercase
   retrieval keywords; include the packet's `task_signature` value as one tag.
   `functions_used` stores observed `app.api` names relevant to the insight; do not
   invent functions.
8. Respect all-failed evidence. If all trajectories failed, produce diagnostic,
   failure, or recovery deltas only. Never invent a success recipe.
9. Keep the output small. At most 3 deltas; one sharp insight beats three generic
   warnings.
# Reflection Focus
The packet's `task_signature` selects the focus block injected below. Mine the named
failure modes first.
{{reflection_focus_block}}
(The pipeline injects exactly one focus block, selected by the packet's
`task_signature`, from the library in `prompts/reflection_focus/*.md`.)
# Grounding and Privacy
- Ground every lesson in outcomes, failed-requirement labels, span sequences, write
  summaries, doc-lookup discipline, or completion shapes present in the packet.
  Unsupported lessons must be omitted.
- Phrase `title`, `applies_when`, and `core_rule` over operation families -- "a
  paginated list API", "a payments/social feed", "a contacts/relationship lookup",
  "a file-tree walk" -- rather than app names. AppWorld apps share engine-wide
  conventions (pagination, login/access_token threading, show_* collections), and
  future tasks involve apps and app combinations not present in these packets.
  Name a specific app only when the lesson is genuinely app-specific and the
  evidence shows it does not hold elsewhere; concrete `app.api` names still belong
  in `functions_used`.
```

Cross-Trajectory Skill Reflector

```
- Do not document API arguments, return schemas, or call syntax; that belongs to
  function cards. Naming which API family the rule concerns is fine.
- Do not output trajectory aliases as if meaningful, task/scenario ids, model names,
  raw evaluation assertions, or copied private text. Never store the task's answer
  or any concrete world value: use `<PERSON>`, `<EMAIL>`, `<PHONE>`, `<ID>`,
  `<PATH>`, `<LOCATION>`, `<DATE_OR_TIME>`, `<AMOUNT>`, `<TITLE_TEXT>`, `<TEXT>`.
- De-identified examples are allowed only to teach a write, recovery, or
  final-answer *shape* -- never actual values.
# Input
task_contrast_packet:
{{task_contrast_packet_json}}
current_relevant_memory:
{{relevant_skill_cards_json}}
# Output
Return exactly one valid JSON object: no markdown, no code fences.
{
  "deltas": [
    {
      "title": "short transferable title",
      "applies_when": "trigger state",
      "solves": "reusable problem",
      "tags": ["keyword", "task_signature_value"],
      "skill": {
        "core_rule": "if-then, ordering, obligation, recovery, or stopping rule",
        "completion_condition": ["done condition"],
        "success_contrast": ["success pattern"],
        "failure_contrast": ["failure pattern"],
        "recovery": ["recovery action"],
        "efficiency": ["efficiency lesson"],
        "procedure_sketch": ["ordered step (text, never code)"]
      },
      "evidence": {
        "observed_feedback": ["failed-requirement or outcome label"],
        "state_transition": ["before -> after"],
        "good_examples": ["good shape, de-identified"],
        "bad_examples": ["bad shape, de-identified"]
      },
      "functions_used": ["app.api"]
    }
  ]
}
```

APPENDIX B.9 | AppWorld Reflector Focus Selection

A deterministic task signature selects exactly one focus block during construction. It changes the evidence emphasized by the shared Reflector prompt; it is neither an ability label nor a runtime gate.

Signature	Primary obligation	Contrasts emphasized
`answer_extraction`	Construct a complete evidence chain and submit the minimal answer value.	Candidate-set coverage, join and aggregation order, temporal anchoring, stopping point, and answer shape.
`bulk_or_multi_write`	Enumerate the full target set and discharge every per-item write exactly once.	Pagination bounds, skipped items, duplicate retries, missing write branches, and ledger closure before finalization.
`state_write`	Resolve the exact target, perform only the requested write, and read back state.	Identity and parameter errors, stale reads, collateral writes, duplicate retries, and premature completion.
`cross_app`	Resolve authoritative cross-app join keys before dependent reads or writes.	Exact-key versus display-name joins, source-of-truth ordering, and propagation of dates, amounts, or identities across apps.
`default`	Use documentation-checked, state-grounded execution and verified finalization.	Skipped documentation, exception recovery, stale variables, operation order, and missing or malformed completion.

Skill-Memory Curator

```
You are a Skill-Memory Curator for an **AppWorld coding agent** (a Python-REPL agent
that calls app APIs as `apis.<app>.<api>(...)` and finishes with
`apis.supervisor.complete_task`).
You receive:
- current relevant skill cards retrieved from the memory bank
- new candidate skill deltas proposed by the Reflector
- compact function cards for observed `app.api` names
Your job is memory editing, not new reflection. Compare each new delta against the
current relevant memory and decide whether it should be added, merged, patched,
narrowed, or ignored. Keep edits local. Do not rewrite the memory bank.
# Curator Protocol
Process every input delta exactly once.
1. Compare the delta with the retrieved cards and choose the smallest valid
   operation.
2. Preserve semantic closure. You may only combine, compress, narrow, or rephrase
   content supported by:
   - the candidate delta,
   - the targeted current cards,
   - the supplied function cards.
   Do not introduce new rules, failure modes, recovery actions, completion
   conditions, procedure steps, examples, or functions.
3. Preserve scope.
   - Function names may appear only in `functions_used`.
   - Do not include API arguments, return schemas, call syntax, or side effects --
     that is function-card territory; a skill card may name the API family a rule
     concerns, nothing lower-level.
   - `procedure_sketch` stays ordered text steps (never code), at most 5.
   - Do not broaden a card merely to absorb a delta. If a delta's evidence applies
     only under a narrower condition than an existing card's trigger, use
     `NARROW_TRIGGER` rather than `PATCH_CARD`.
4. Preserve privacy, genericity, and compactness. Cards must remain free of concrete
   world values (people, titles, amounts, ids, dates), task answers, task/scenario
   ids, and model names; keep placeholders as-is (`<PERSON>`, `<EMAIL>`, `<PHONE>`,
   `<ID>`, `<PATH>`, `<LOCATION>`, `<DATE_OR_TIME>`, `<AMOUNT>`, `<TITLE_TEXT>`,
   `<TEXT>`). Produce a concise resulting card; keep `tags` 3-8 items and
   retain any `task_signature` tags supported by delta or targets.
# Operation and Patch Rules
- `ADD_CARD`: no semantically equivalent card exists. Provide the complete new card.
- `PATCH_CARD`: the delta adds supported content without materially changing card
  scope. Provide the complete updated card.
- `MERGE_CARD`: two or more existing cards express substantially the same
  transferable rule. Include all involved cards in `target_card_ids`, set
  `target_card_id` to the retained canonical card, and provide the complete merged
  card.
- `NARROW_TRIGGER`: the existing rule is useful but `applies_when` is too broad
  (e.g., a pagination-exhaustion rule firing on single-entity lookups). Provide the
```

```
  complete narrowed card.
- `NOOP`: the delta adds no meaningful information, its evidence is weak or
  conflicting, or it is only API syntax (function-card territory).
  `new_or_updated_card` must be null.
# Input
current_relevant_skill_cards:
{{current_relevant_skill_cards_json}}
new_deltas:
{{skill_deltas_json}}
related_function_cards:
{{related_function_cards_json}}
# Output
Return exactly one valid JSON object. No markdown. No code fences.
{
  "patches": [
    {
      "delta_index": 0,
      "operation": "ADD_CARD",
      "target_card_id": "existing skill::skill_<hex> id or null",
      "target_card_ids": ["skill::skill_<hex>"],
      "new_or_updated_card": {
        "title": "short transferable title",
        "applies_when": "trigger state",
        "solves": "reusable problem",
        "tags": ["keyword", "task_signature_value"],
        "skill": {
          "core_rule": "if-then, ordering, obligation, recovery, or stopping rule",
          "completion_condition": ["done condition"],
          "success_contrast": ["success pattern"],
          "failure_contrast": ["failure pattern"],
          "recovery": ["recovery action"],
          "efficiency": ["efficiency lesson"],
          "procedure_sketch": ["ordered step (text, never code)"]
        },
        "evidence": {
          "observed_feedback": ["feedback label"],
          "state_transition": ["before -> after"],
          "good_examples": ["good shape, de-identified"],
          "bad_examples": ["bad shape, de-identified"]
        },
        "functions_used": ["app.api"]
      },
      "reason": "one concise evidence-grounded reason"
    }
  ]
}
```

Static Task-Start Injection (C1a)

```
APPWORLD STATIC TASK-START MEMORY INJECTION (C1a)
PURPOSE
The renderer fills this template deterministically from retrieved cards. No LLM
writes the block. The official AppWorld ReAct prompt, worked example, execution
shell, and Supervisor.complete_task contract remain unchanged.
PLACEMENT
Insert the block inside the final USER message of the official
react_code_agent prompt, after the Key Instructions and immediately before:
  Using these APIs, now generate code to solve the actual task:
If retrieval selects no cards, insert nothing. Under the reported C1a condition,
only this initial USER message differs from no memory; all later messages are
structurally identical. The bank is frozen throughout evaluation.
BUDGET
- Hard cap: 1,200 tokens (a cap, not a target).
- At most 3 Skill Cards, each at most 140 rendered tokens.
- At most 2 Function/API Cards, each at most 150 rendered tokens.
- No example dialogue is inserted.
TASK-START BLOCK TEMPLATE
## Retrieved Procedural Memory (soft guidance)
These notes were distilled offline from previous agents' successes and failures
on other tasks. They are soft guidance about procedure, not facts about the
current world. Live API docs (apis.api_docs) and current execution outputs are
always the ground truth.
Skill memory (task start):
1. [skill] {{title}} - {{core_rule}} Done when:
   {{completion_condition_joined}}.
   {{optional: Pitfall: failure_contrast_top1.}}
   {{optional: Recovery: recovery_top1.}}
   {{optional: Procedure: procedure_sketch as "a -> b -> c".}}
2. [skill] ...
3. [skill] ...
Function memory (likely-relevant APIs):
- Tool memory for {{app.api}}: {{what_it_does_short}}
  {{usage_rules_top2_joined}}
  {{optional: Known mistakes: common_mistakes_top2.}}
  {{optional: Idiom (adapt, never paste): code_idiom_one_line.}}
- Tool memory for {{app.api}}: ...
Memory-use rules:
- Apply a memory only when its trigger matches what you actually observe in this
```

```
  task; if memory and a live observation or API doc disagree, trust the live one.
- Never reuse concrete values from memory (ids, tokens, emails, names, dates,
  amounts). Look every value up through APIs in this session.
- Memory does not replace documentation discipline: still call
  apis.api_docs.show_api_doc before the first use of any API.
- Adapt idioms to your current variables; never paste them verbatim.
- Skill memory decides when or why to read, write, verify, or stop; Function
  memory only explains how a specific API behaves.
- Before any state-changing write, confirm the exact target set from fresh reads.
  After the last required write, verify by reading the state back before calling
  apis.supervisor.complete_task.
- For answer tasks, stop exploring once the answer is determined; submit the
  minimal value (numbers numeric, not words).
- Do not repeat unbounded identical reads; after repeated empty or unchanged
  results, change strategy.
- Applying a skill must change the next code cell. For example, a pagination
  skill means looping page_index until an empty page rather than counting one
  call's result.
RENDERING
- Strip optional segments whose fields are empty.
- Join list fields with "; ".
- Keep one line per card where possible.
- Collapse code_idiom to one semicolon-separated line only when it is at most
  90 characters; otherwise render a short pattern description.
RETRIEVAL INPUT CONTRACT
{
  "task_instruction": "...",
  "apps_in_instruction": ["spotify"],
  "runtime_signature": "answer_extraction",
  "skill_topk": 3,
  "function_topk_start": 2,
  "exclude_generator_ids": ["..."],
  "render_code_idiom": true,
  "jit": {"enabled": false}
}
The renderer must never emit task, scenario, or trajectory ids; source-model
names; card provenance; raw retrieval-score components; or concrete world values
carried in offline evidence fields.
```