EVOTRACE: BENCHMARKING SELF-EVOLVING AGENTS ON PROCEDURAL TASKS THROUGH EXPERIENCE-TO-BEHAVIOR TRACING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents deployed in professional domains repeatedly encounter requests governed by the same underlying procedure, where a missed operation, conditional decision, or dependency can invalidate an execution. For self-evolving agents that seek to improve from experience, effective evolution requires extracting reusable procedural knowledge from prior task interactions and correctly applying it to new requests. However, existing benchmarks mainly reveal aggregate performance changes or the quality of retained knowledge, leaving it unclear which procedural requirements are actually learned and where the experience-to-behavior pathway breaks down. We introduce EVOTRACE, a benchmark that uses Agent-Executable Standard Operating Procedures (AE-SOPs) as a common reference to track each procedural requirement across actual experience, explicit retention, and held-out execution. EVOTRACE comprises 115 AE-SOPs across seven professional domains and 10,792 executable instances. Experiments across *eleven* representative self-evolution methods show that effective evolution cannot be inferred from either the form or the coverage of retained knowledge alone: similar forms can accompany substantially different gains, while broader coverage does not necessarily translate into more reliable execution. Instead, different mechanisms exhibit distinct bottlenecks in how procedural requirements progress through the experience-to-behavior pathway, and these vulnerabilities vary across requirement types. Targeting the identified bottlenecks with lightweight interventions improves subsequent execution.

1 INTRODUCTION

Large language model (LLM) agents are increasingly deployed for multi-step tasks in professional domains such as customer service and software engineering (Drouin et al., 2024; Yang et al., 2024). In many of these settings, agents repeatedly encounter requests that can be handled using the same underlying procedure, despite differences in task details and user wording. The procedure specifies all *procedural requirements*, including operations, conditional decisions, and their dependencies; together, these requirements constitute a Standard Operating Procedure (SOP) (U.S. Environmental Protection Agency, 2007). However, such SOPs are not always supplied to agents and may no longer apply as an agent's operating environment changes (Jiang et al., 2026; Lin et al., 2026).

Self-evolving agents seek to abstract reusable procedures from their experience, retain them in persistent artifacts, and apply them to improve subsequent execution (Zhao et al., 2024; Zhang et al., 2026a; Alzubi et al., 2026). For illustration, consider a customer-service agent handling transaction reversals (Figure 1). The SOP requires the agent to inspect the payment state, choose between a refund and an authorization cancellation, obtain user confirmation, and perform the appropriate update (PayPal, 2026). Yet a single execution trace mixes reusable procedural requirements with task-specific details, and typically exposes only a subset of these requirements; for example, the observed payment state may activate only the refund branch. Therefore, effective self-evolution should integrate partial evidence accumulated across experiences, retain reusable knowledge in artifacts, and apply the inferred procedure to new requests that are potentially governed by the same SOP.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Existing self-evolution mechanisms pursue this objective through different forms of persistent knowledge, including retrieved task cases (Zhou et al., 2025), reusable knowledge units (Ouyang et al., 2026), iteratively revised guidance (Zhao et al., 2024; Zhang et al., 2026a), and modular skills (Alzubi et al., 2026). Current benchmarks primarily conduct end-to-end evaluations and produce a final score on whether accumulated experience improves subsequent task performance or transfers to new tasks (Wei et al., 2025; Lei et al., 2026); some additionally examine the quality or reuse of the retained knowledge (Zhong et al., 2026).

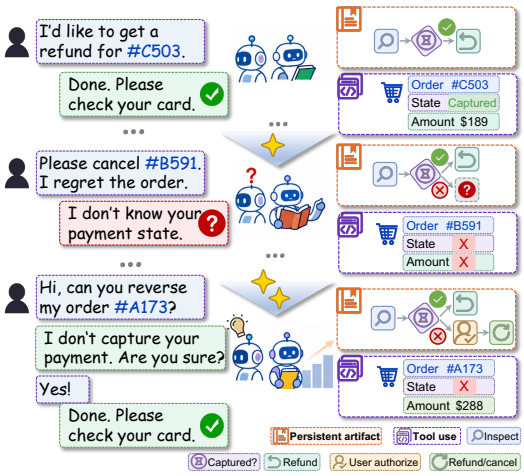


Figure 1: Self-evolving agent learns reusable procedure from execution experiences.

However, these evaluations do not establish procedural requirement-level correspondence across the full learning process: whether a specific procedural requirement was actually encountered in experience, indeed retained in persistent knowledge, and effectively realized in behavior. This missing correspondence limits what can be inferred from aggregate performance changes: *which* procedural requirements fail to carry through the evolution process and *where* along the pathway the failure occurs. For example, a missed procedure realization may arise from insufficient exposure, failed retention, or failed reuse; across mechanisms, those with similar overall gains may suffer from distinct vulnerabilities, such as poor retention of dependency requirements or poor realization of conditional-decision requirements. This motivates a central evaluation question: *How well does a self-evolving agent acquire reusable procedural knowledge from accumulated experience and apply it to unseen instances governed by the same procedure?*

To track the experience-to-behavior pathway at the requirement level, we introduce Agent-Executable Standard Operating Procedure (AE-SOP). An AE-SOP specifies all procedural requirements verifiable from execution traces, establishing a common reference for requirement-level correspondence across experience traces, persistent artifacts, and held-out executions. EvoTrace provides 115 AE-SOP families drawn from six procedural sources across seven application domains, along with modules to support the extension (Figure 3). Each family provides an AE-SOP reference and executable instances that an agent actually encounters (10,792 in total), all governed by the same procedure. For each evaluated method, EvoTrace provides a unified evaluation interface: it presents instances to the agent through simulated user interactions and supports execution with the corresponding tools in a sandboxed environment. The agent handles these instances and evolves through its native mechanism, retaining reusable procedural knowledge from this experience in persistent artifacts. EvoTrace then evaluates evolution effectiveness by freezing the agent in this evolved state and having it handle held-out instances.

Using EvoTrace, experiments across *eleven* representative self-evolution methods yield several key insights. First, broader retention or coverage of procedural requirements does not by itself ensure more reliable held-out execution. Second, no evaluated knowledge representation or mechanism category consistently guarantees effective evolution: methods that persist knowledge in similar forms can yield substantially different gains and exhibit distinct vulnerabilities across learning stages and requirement types. Third, these pathway-level diagnostics are actionable: once a bottleneck is localized, lightweight interventions targeted at retention or execution-time reuse can improve subsequent procedural execution. Together, these findings show that aggregate outcomes or retained knowledge alone are insufficient to characterize self-evolution; effective improvement requires both knowledge-type adaptation and learning-stage optimization.

Our contributions are: (1) we propose an AE-SOP-grounded evaluation framework that establishes requirement-level correspondence across experience, persistent artifacts, and held-out execution; (2) we open-source a dataset comprising 115 AE-SOPs across seven application domains and 10,792 experience and held-out instances, together with a reusable construction and evaluation pipeline; and (3) we conduct a systematic empirical study of eleven representative self-evolution methods, showing their distinct evolution bottlenecks that can guide targeted improvements.

2 Background and Related Work

2.1 SOP and Procedural Execution

Standard Operating Procedures (SOPs) are documented, standardized workflows for ensuring consistent execution of a family of tasks across domains such as manufacturing, healthcare, and administration (U.S. Environmental Protection Agency, 2007). As LLM agents are increasingly deployed in tool-using workflows, SOPs provide a natural specification of the rules, decisions, and actions that should govern agent behavior. The τ family combines domain policies, tool use, and agent–user coordination in customer-service tasks (Yao et al., 2024; Barres et al., 2025). SOPBench evaluates action permissibility of required checks through executable procedural constraints (Li et al., 2025), while Amazon SOP-Bench evaluates agents on expert-authored industrial procedures with branching logic and diverse tools (Nandi et al., 2025). SWE-Skills-Bench measures the utility of supplied skill documents through paired evaluations with and without the corresponding skills (Han et al., 2026). Together, these benchmarks characterize an agent's ability to execute or benefit from procedural knowledge available at inference time. However, such procedures cannot always be directly supplied to the agent: Human-authored SOPs may not apply to agent's observations, tools, and action space; moreover, changes to tools or the underlying environment can render previously valid procedures outdated (Jiang et al., 2026; Lin et al., 2026). Under this scenario, a deployed agent needs to explore reusable procedural knowledge from its own task interactions.

2.2 Self-Evolving Agents and Benchmarks

Experience-driven self-evolving agents provide mechanisms for acquiring and maintaining procedural knowledge. They accumulate reusable information from prior interactions in persistent external artifacts, enabling subsequent performance improvement without retraining the backbone LLM (Gao et al., 2026). Existing approaches differ primarily in the form of knowledge they persist and reuse. *Case memories* preserve concrete execution traces for retrieval as demonstrations on later tasks (Zhou et al., 2025; Xu et al., 2025) *Knowledge banks* abstract experience into reusable knowledge units that can be progressively refined and retrieved across tasks (Ouyang et al., 2026; Fang et al., 2026; Cao et al., 2026). *Guidance documents* maintain textual strategies and supply them as guidance for subsequent execution (Zhao et al., 2024; Suzgun et al., 2026; Zhang et al., 2026a; Wang et al., 2025). *Skill libraries* package procedural knowledge into modular skills that can be revised, retrieved, and composed (Alzubi et al., 2026; Zhou et al., 2026).

Benchmarks for self-evolution evaluate this process from several complementary perspectives. *Outcome-oriented evaluation* measures whether accumulated experience improves subsequent task success, efficiency, or transfer (Wei et al., 2025; Wang et al., 2026; Belikova et al., 2026; Fu et al., 2026; Lei et al., 2026). *Artifact-oriented evaluation* examines the quality, coverage, or utility of the memories and skills produced through evolution (Zhang et al., 2026b; Zhong et al., 2026). *Execution-time evaluation* analyzes their reuse in subsequent execution, including skill usage and procedural completion (Zhang et al., 2026b; Zhong et al., 2026). These evaluations reveal whether evolution improves retained artifacts or subsequent behavior, but they do not establish a common requirement-level correspondence across the full learning pathway.

3 The EvoTrace Benchmark

3.1 Benchmark Objective and Design

Objective. EvoTrace tracks how self-evolving agents acquire reusable procedural knowledge from accumulated task experience and apply it to unseen instances governed by the same procedure. To make this learning pathway measurable, we track individual procedural requirements across three stages: (1) whether the agent encounters each requirement during task interactions, (2) whether it explicitly retains that requirement in persistent artifacts, and (3) whether it correctly realizes the requirement on held-out instances governed by the same procedure. Tracking these observations as experience accumulates reveals where learning succeeds or breaks down and how learning pathways differ across evolution mechanisms. The key challenge is to establish correspondence for the *same procedural requirements* across these stages, even though each execution trace activates only the partial branches and mixes reusable requirements with instance-specific details.

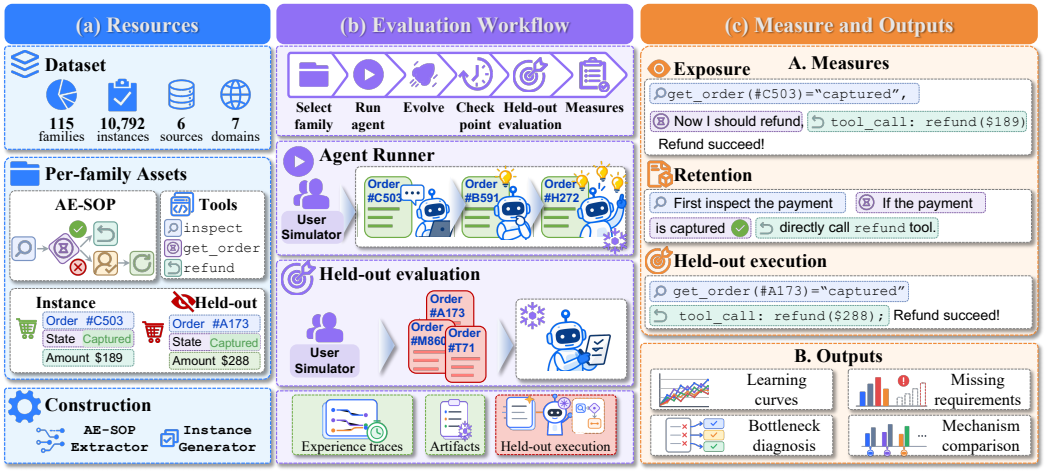


Figure 2: EvoTrace Overview.

Design. To establish this correspondence, EvoTrace associates each task family with an **Agent-Executable Standard Operating Procedure (AE-SOP)**: a graph specifying required operations, conditional decisions, and dependencies. *Standard* denotes the shared, instance-invariant procedure that governs tasks with different details. *Agent-executable* means that the agent can perform the required operations using available tools and make decisions based on its observations; this reference can therefore be used to assess, from execution traces, whether the agent fulfills each applicable requirement. The AE-SOP characterizes the ground-truth execution of the task family, and serves as the common procedural reference in evaluation, while remaining hidden from the agent.

Each AE-SOP governs a set of task **instances** that an agent actually handles, each comprising a task request, relevant entities and attribute values, and any required user feedback. Within each family, a subset is reserved for *held-out* evaluation on unseen task realizations, while the remaining instances are presented to the agent during evolution, providing the task-solving experience from which it may learn. In the transaction-reversal example (Figure 1), instances vary in order identifiers, payment states, and request wording, while the same AE-SOP determines which branch and operations apply. Tracking the learning process requires examining runtime evidence collected throughout evolution. For each evolution checkpoint, this involves accumulated traces serving as task-solving experiences, snapshots of the persistent artifacts, and execution traces on held-out instances. These evidence sources capture what the agent encountered, what reusable procedural knowledge it retained, and how its evolved state behaved on unseen tasks, respectively; aligning them with the same AE-SOP connects actual exposure, explicit retention, and procedural execution of the same requirements.

3.2 Benchmark Resources and Usage

Figure 2 summarizes the resources and evaluation workflow.

Benchmark Resources. EvoTrace provides a benchmark dataset together with reusable components for dataset extension and evaluation. Figure 3 presents the dataset profile, comprising 115 AE-SOP families with 10,792 instances from six procedural sources across seven application domains. Each family includes an AE-SOP reference, its task instances with a held-out split, and the supporting tools and execution environment (details in Appendix 5). For dataset extension, EvoTrace provides the `AE-SOP Extractor` and `Instance Generator` (Figure 2(a)). The evaluation components include the `Agent Runner`, which manages task sessions and

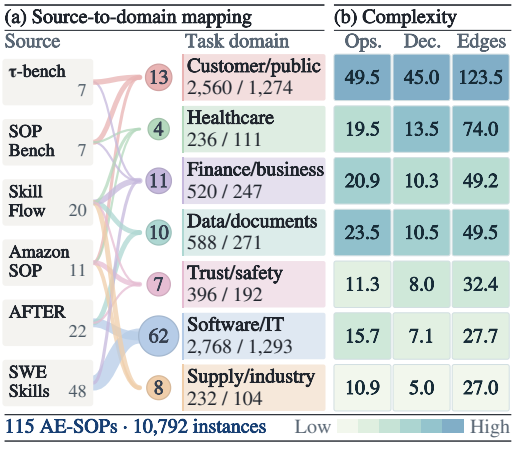


Figure 3: EvoTrace Dataset Profile.

checkpoint evaluation, with the `Adaptive User Simulator` supplying task-dependent requests and responses (Figure 2(b)). The `Metric Engine` aligns experience traces, artifact snapshots, and held-out execution traces with the AE-SOP reference to compute requirement-level measures along the learning pathway (Figure 2(c)).

Evaluation Workflow. Researchers select existing families or extend the dataset using `AE-SOP Extractor` and `Instance Generator` (§ 4). After specifying evolution checkpoints, the `Agent Runner` executes experience instances in the corresponding task environments, with the `Adaptive User Simulator` supplying initial requests and required user responses based on the task context. The agent updates its persistent artifacts through its native evolution mechanism. At each checkpoint, the runner snapshots the current evolution state and evaluates independent frozen copies on the same held-out set. Test interactions remain isolated from subsequent evolution. The runner passes the cumulative experience traces, the artifact snapshot, and the held-out execution traces to the `Metric Engine`, which reuses the extractor for artifact analysis and aligns all three evidence sources with the reference. Tracking these measures across checkpoints connects experience with explicit retention and procedural execution, supporting comparison of learning pathways (§ 5).

4 Benchmark Construction

4.1 Grounding Verifiable Procedural References

Sources and Admission Criteria. We select procedural sources according to three criteria: (1) their procedures apply to multiple task instances, (2) they provide tools and environments to support agents in executing the required operations, and (3) they provide verifiers that assess execution against procedural requirements using observable runtime evidence. The six sources comprise domain policies from the τ benchmark family (Yao et al., 2024), dependency-constrained service procedures from SOPBench (Li et al., 2025), industrial SOP documents from Amazon SOP-Bench (Nandi et al., 2025), and reusable execution flows from SkillFlow (Zhang et al., 2026b), AFTER (Belikova et al., 2026), and SWE-Skills-Bench (Han et al., 2026). Appendix B details their procedural materials, execution support, verification scope, and adaptations.

AE-SOP Reference Grounding. For each AE-SOP family $\mathcal{P}$, we ground its procedural graph $G_{\mathcal{P}} = (V_O, V_D, E)$ in a shared execution environment. Operation nodes in V_O group tool invocations achieving independently observable subgoals; decision nodes in V_D use tool evidence or user facts to select branches; and directed edges in E encode necessary data or prerequisite dependencies rather than textual order. Each graph element $z \in V_O \cup V_D \cup E$ identifies an operational atom $a_z = \langle z, \lambda_z, \eta_z \rangle$, where λ_z specifies its activation conditions, and η_z specifies the observable evidence needed to assess its realization in an execution trace. This evidence may include tool inputs and outputs, state changes, user feedback, and constraints on event ordering, repetition, or termination. Each atom also retains source evidence $\rho(z)$ supporting its procedural claim. The annotated graph yields the reference set $\mathcal{Z}_{\mathcal{P}}^{\text{ref}} = \{a_z : z \in V_O \cup V_D \cup E\}$.

Extraction and Validation. We deploy a LLM-based `AE-SOP Extractor` to map procedural sources into the references and validate them in four stages. (1) Three backbone LLMs independently extract graph elements; candidates require semantically equivalent proposals from at least two models (Elumar et al., 2025). (2) An independent verifier agent checks each claim against $\rho(z)$, escalating insufficient support and conflicting interpretations for expert review (Min et al., 2023). (3) We execute source-provided test cases and use source-native verifiers to identify successful runs, corroborating the candidate requirements exercised in those runs. (4) Three human experts independently review 20% of graph samples covering all sources and complexity levels. Table 1(a) reports the validation results. Appendix C provides annotations, extractions, and review details.

4.2 Generating Executable Instance Variations

Branch-Conditioned Generation. For each AE-SOP, we specify target requirements and use the `Instance Generator` to trace their branch conditions and prerequisite dependencies backward, identifying consistent task facts and user responses. Starting from source-provided runnable instances where available, it generates concrete requests, entity configurations, and response specifications satisfying these constraints. For instance x, these facts and branch conditions determine the applicable reference subset $\mathcal{Z}_x^{\text{ref}} \subseteq \mathcal{Z}_{\mathcal{P}}^{\text{ref}}$.

Table 1: Benchmark-validation results. All results are reported as percentages.

(a) Reference validation

Measure	Result
Extraction agreement	89.0
Claim–evidence support	90.3
Trajectory corroboration	88.2
End-to-end corroboration	70.8
Expert-review agreement	82.6
Reference completeness	86.9

(b) Instance / split validation

Measure	Result
Configuration acceptance	84.0
Linguistic consistency	91.8
Entity-identifier overlap	0.0
Exact-template overlap	0.0
Human instance agreement	89.2

(c) Matching / scoring validation

Measure	Result
Matching precision	91.4
Matching recall	83.7
Requirement agreement	81.3
Workflow-label agreement	84.2
Human scoring agreement	80.3

Linguistic Variation and Partitioning. For each task entity configuration, LLMs generate alternative linguistic realizations of its initial request and user responses while preserving task facts and required response content. Within each targeted $\mathcal{Z}_x^{\mathrm{ref}}$, we partition instances into experience and test sets with disjoint entity identifiers and linguistic templates. The split supports testing the shared procedural requirements with new entities and language.

Instance and Split Validation. Before linguistic variation, we execute each configuration to verify executability and agreement between its activated requirements and $\mathcal{Z}_x^{\mathrm{ref}}$. We then check factual and response-content consistency after variation, and audit entity-identifier and linguistic-template overlap, as well as $\mathcal{Z}_x^{\mathrm{ref}}$ coverage after splitting. Table 1(b) summarizes these checks; Appendix D details branch-conditioned generation, linguistic variation, split construction, and validation. Requirement coverage describes the learning opportunities provided by the dataset; an agent's actual requirement exposure is assessed separately from its acquisition traces (§ 5.1).

5 Evaluation Protocol and Measures

5.1 Evaluation Protocol

Interaction and Information Boundaries. Each instance runs in a fresh session and isolated environment; only persistent evolution state carries between experience instances. `Adaptive User Simulator` supplies the initial task request and attachments, then releases feedback according to a disclosure policy based on user-observable task context, while tools return native outputs and errors. Reference AE-SOPs and verification annotations remain inaccessible to the agent. At checkpoint c_t, after n_t experience instances, the complete evolving state is evaluated on the fixed held-out set $\mathcal{X}^{\mathrm{test}}$. Each test instance runs in a reset environment from an independent frozen copy. Neither test interactions nor evaluation outputs enter other test instances or subsequent evolution. Appendix E specifies disclosure and isolation rules.

Evidence Processing and Requirement Matching. At each checkpoint, we collect an evidence package containing cumulative experience traces T_t^{train}, an artifact snapshot A_t, and held-out execution traces $T_t^{\mathrm{test}} = \{\tau_{t,x} : x \in \mathcal{X}^{\mathrm{test}}\}$. The `Metric Engine` extracts atoms $\widehat{\mathcal{Z}}_t^{\mathrm{train}}$ from agent-visible experience evidence and test-execution atoms $\widehat{\mathcal{Z}}_{t,x}^{\mathrm{test}}$ from each $\tau_{t,x}$, and reuses the `AE-SOP Extractor` to recover explicitly encoded atoms $\widehat{\mathcal{Z}}_t^{A}$ from A_t. We use $\cap$ to denote evidence-grounded matching of equivalent requirements between two sets. Such equivalence is determined by deterministic checks that assess runtime evidence under λ_z and η_z, and semantic matching that relates procedural claims across wording and instance-specific values. Appendix F details the matching protocol; Table 1(c) reports its validation.

5.2 Metric System

Procedural Execution. *Final Success Rate (FSR)* measures the fraction of test instances whose traces satisfy all applicable requirements. *Atom Manifestation Rate (AMR)* first measures the fraction of applicable requirements correctly realized in each trace, then averages these fractions equally:

$$\mathrm{FSR}_t = \frac{\left|\left\{x \in \mathcal{X}^{\mathrm{test}} : \widehat{\mathcal{Z}}_{t,x}^{\mathrm{test}} \cap \mathcal{Z}_x^{\mathrm{ref}} = \mathcal{Z}_x^{\mathrm{ref}}\right\}\right|}{|\mathcal{X}^{\mathrm{test}}|}, \quad \mathrm{AMR}_t = \frac{1}{|\mathcal{X}^{\mathrm{test}}|} \sum_{x \in \mathcal{X}^{\mathrm{test}}} \frac{\left|\widehat{\mathcal{Z}}_{t,x}^{\mathrm{test}} \cap \mathcal{Z}_x^{\mathrm{ref}}\right|}{|\mathcal{Z}_x^{\mathrm{ref}}|}. \quad (1)$$

Explicit Retention. We assess retained knowledge against the agent's actual experience and the complete procedural reference. *Knowledge Precision (KP)* measures the fraction of encoded atoms supported by the reference; *Knowledge Recall (KR)* measures the fraction of observed requirements retained in artifacts; and *Knowledge Coverage (KC)* measures coverage of the complete reference:

$$\mathrm{KP}_t = \frac{|\widehat{\mathcal{Z}}_t^A \cap \mathcal{Z}^{\mathrm{ref}}|}{|\widehat{\mathcal{Z}}_t^A|}, \quad \mathrm{KR}_t = \frac{|\widehat{\mathcal{Z}}_t^A \cap \widehat{\mathcal{Z}}_t^{\mathrm{train}} \cap \mathcal{Z}^{\mathrm{ref}}|}{|\widehat{\mathcal{Z}}_t^{\mathrm{train}} \cap \mathcal{Z}^{\mathrm{ref}}|}, \quad \mathrm{KC}_t = \frac{|\widehat{\mathcal{Z}}_t^A \cap \mathcal{Z}^{\mathrm{ref}}|}{|\mathcal{Z}^{\mathrm{ref}}|}. \tag{2}$$

Retention–Execution Correspondence. *Knowledge-to-Behavior (K2B)* similarly averages, within each eligible trace, the fraction of retained applicable requirements that are correctly realized:

$$\mathrm{K2B}_t = \frac{1}{\left|\left\{x \in \mathcal{X}^{\mathrm{test}} : |\widehat{\mathcal{Z}}_t^A \cap \mathcal{Z}_x^{\mathrm{ref}}| > 0\right\}\right|} \sum_{\substack{x \in \mathcal{X}^{\mathrm{test}} \\ |\widehat{\mathcal{Z}}_t^A \cap \mathcal{Z}_x^{\mathrm{ref}}| > 0}} \frac{|\widehat{\mathcal{Z}}_t^A \cap \widehat{\mathcal{Z}}_{t,x}^{\mathrm{test}} \cap \mathcal{Z}_x^{\mathrm{ref}}|}{|\widehat{\mathcal{Z}}_t^A \cap \mathcal{Z}_x^{\mathrm{ref}}|}. \tag{3}$$

We track these measures across checkpoints against the number of experience instances to characterize the learning dynamics. Appendix G.3 defines key geometric properties in the learning curves. Appendix H analyzes the execution profiles hidden by aggregate scores, the relation between AMR and FSR, and the effect of changing retention on K2B.

6 Experiment

Using EvoTrace, we answer three questions about *eleven* representative self-evolution methods, progressing from execution outcomes to bottleneck diagnosis and targeted intervention: (i) How does procedural execution progress as experience accumulates?(§ 6.2) (ii) Where do bottlenecks arise across requirement types and learning stages? (Appendix L) (iii) Can trace-guided interventions improve procedural execution? (Appendix M) We additionally examine robustness, learning efficiency, and measurement reliability (Appendices N–P). Appendices additionally provide implementation details (J), metric statistics (K), extended diagnostics (L), targeted interventions (M), robustness and sensitivity (N), computational cost (O), and measurement reliability (P).

6.1 Experimental Setup

Methods and Baselines. We evaluate *eleven* self-evolution methods to cover representative persistent-representation designs that shape how procedural knowledge is abstracted, retained, and reused: *case memories* (Memento-NP and A-Mem) (Zhou et al., 2025; Xu et al., 2025); *knowledge banks* (ReasoningBank, MemP-online, and ReMe) (Ouyang et al., 2026; Fang et al., 2026; Cao et al., 2026); *guidance documents* (ExpeL, Dynamic Cheatsheet, ACE, and AWM-online) (Zhao et al., 2024; Suzgun et al., 2026; Zhang et al., 2026a; Wang et al., 2025); and *skill libraries* (EvoSkill and Memento-Skills) (Alzubi et al., 2026; Zhou et al., 2026). We include *two baselines*: *No update* that does not evolve, and *trace retrieval*, which retrieves prior execution traces using cosine similarity between task-description embeddings from a fixed encoder.

Backbone Selection. We use GPT-5.6-Terra (OpenAI, 2026b) and open-weight Qwen3.8-Flash-Next (Qwen Team, 2026b) as primary backbones for two reasons: (1) both are positioned by their developers to balance capability and inference cost, which is suitable for sustained agent deployment; (2) we follow prior self-evolution studies (Ouyang et al., 2026; Zhang et al., 2026a), choosing contemporary models that occupy similar capability–cost tiers within leading model families.

Evaluation Settings. We select 60 AE-SOP families following the dataset distribution. For each SOP family, we repeat evolution under *three* shuffled instance orders as experience; each run starts with a reset state. Within each repetition, all methods share the instance order and held-out test. We evaluate at initialization (0%) and after 25%, 50%, 75%, and 100% of the experience instances, evaluating the respective evolution checkpoints.

6.2 Procedural Execution over Evolution

Table 2 reports held-out FSR/AMR performance and their changes from initial state after all experience instances. Averaged across the eleven evolving methods, FSR improves by approximately 8.8

and 11.9 percentage points (pps) on GPT and Qwen, respectively. The comparison between AMR and FSR reveals that high realization of requirements may still leave the whole procedure execution fail: ACE reaches 92.8% AMR with 65.0% FSR on GPT. Methods with the same evolution artifact type can differ substantially, where ReasoningBank and MemP differ in FSR gain (11.1 versus 1.1 pp). These contrasts illustrate that requirement-level correctness and artifact category do not determine success of complete held-out execution.

Table 2: Overall performance on EvoTrace (%); gains (Δ) are Percentage Points (pp).

		GPT-5.6-Terra				Qwen3.8-Flash-Next			
Class	Method	FSR	ΔFSR	AMR	ΔAMR	FSR	ΔFSR	AMR	ΔAMR
Ctrl	No upd.	45.5	-1.3	86.0	-0.1	47.3	+1.5	86.5	+0.6
Ctrl	Retrieval	48.2	+1.4	87.2	+1.1	50.7	+4.9	87.9	+1.9
CN	Memento	53.4	+6.6	88.9	+2.8	56.0	+10.2	89.8	+3.9
CN	A-Mem	49.8	+3.0	87.6	+1.5	52.4	+6.7	88.4	+2.5
KB	RBank	57.9	+11.1	90.8	+4.6	59.7	+13.9	91.3	+5.4
KB	MemP	47.9	+1.1	86.9	+0.7	49.9	+4.1	87.4	+1.4
KB	ReMe	57.3	+10.5	90.6	+4.4	58.4	+12.6	90.8	+4.9
GW	ExpeL	57.8	+11.0	90.8	+4.6	60.1	+14.3	91.2	+5.3
GW	Cheatsheet	47.5	+0.7	86.6	+0.4	49.3	+3.6	87.3	+1.3
GW	ACE	65.0	+18.2	92.8	+6.6	67.2	+21.4	93.3	+7.4
GW	AWM	55.5	+8.7	90.0	+3.8	58.3	+12.5	90.7	+4.7
SL	EvoSkill	59.6	+12.9	91.3	+5.2	61.0	+15.2	91.5	+5.6
SL	M-Skills	59.9	+13.1	91.4	+5.2	62.1	+16.3	92.0	+6.0
Evolving Avg (11)		55.6	+8.8	89.8	+3.6	57.7	+11.9	90.3	+4.4

Ctrl: baselines; CN: case memories; KB: knowledge banks; GW: guidance documents; SL: skill libraries.

Figures 4 and 5 track held-out execution and explicit knowledge coverage across checkpoints. Higher knowledge coverage does not consistently correspond to larger execution gains across methods: Dynamic Cheatsheet reaches 49.4% KC with a 0.7 pp FSR gain, whereas EvoSkill reaches 13.5% KC with a 12.9 pp gain. This contrast motivates examining which requirements were encountered, retained, and realized, rather than interpreting artifact coverage alone as successful learning.

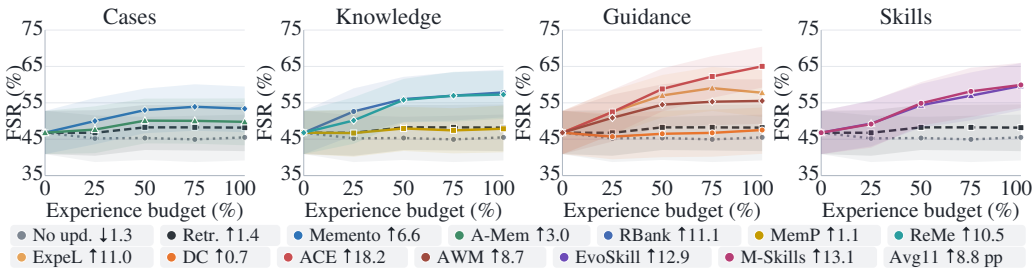


Figure 4: Final Success Rate (FSR) curves on held-out tasks.

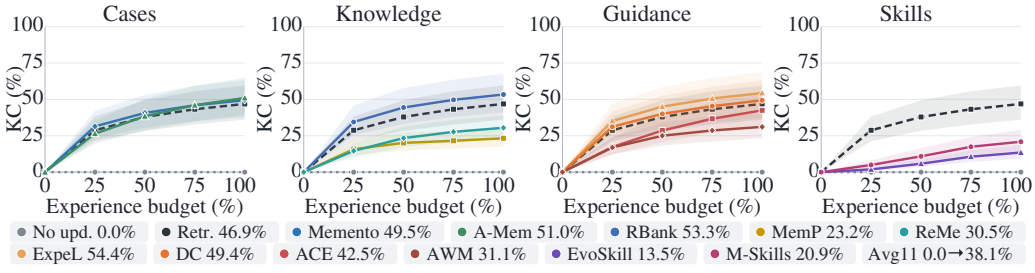


Figure 5: Knowledge Coverage (KC) over accumulated experience.

6.3 Learning Bottlenecks Diagnostics

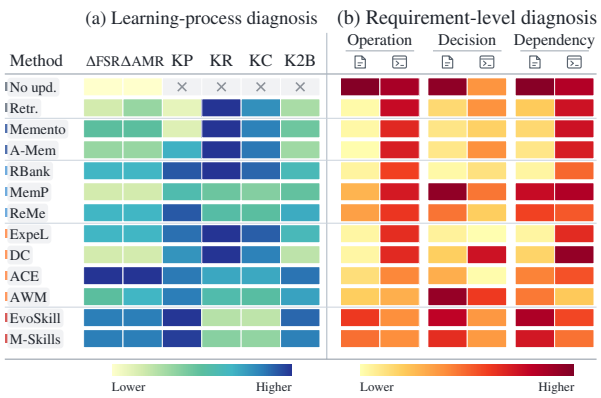


Figure 6: Evolution pathway diagnostics. persistent artifact; execution.

Figure 6 relates these outcomes with process-level metrics, then separates operation, decision, and dependency misses among failed execution. Retaining observed requirements does not imply coverage, while higher coverage alone does not guarantee substantial execution gains. Dynamic Cheatsheet achieves 100.0% KR on GPT, yet covers only 49.4% of reference requirements and improves FSR by just 0.7 pp. Despite its higher KC than EvoSkill (13.5%), its FSR gain is much smaller (0.7 vs. 12.9 pp). The type-level breakdown on failed executions further reveals each mechanism's vulnerability to certain types of requirements: decision knowledge misses in artifacts occur in 63.7% of MemP-online's failed executions, compared with 8.5% for ReasoningBank. K2B has the strongest Pearson correlation with ΔFSR among the four process measures (0.890 on GPT and 0.896 on Qwen). This association suggests that correctly realizing retained requirements is most closely related to the final improvement.

6.4 Trace-Guided Interventions

Motivated by the diagnostic profiles, we examine two stage-targeted, lightweight modifications to improve the corresponding process metrics and held-out FSR. For retention, we use three-model voting to select candidate knowledge for the persistent artifact, which on average increases KC by 7.79 pp and FSR by 6.31 pp on targeted methods with GPT. For execution-time reuse, we add an independent applicability review from top-3 retrieved knowledge, leading to average increase of 0.89 pp K2B and 5.85 pp FSR on targeted methods with GPT. Figure 7 demonstrates these effects.

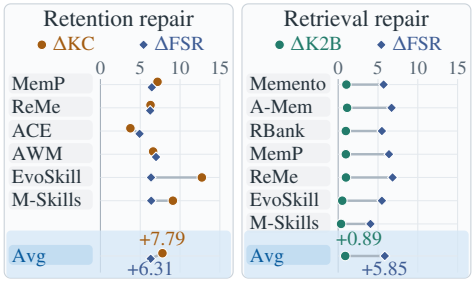


Figure 7: Targeted repair gains.

6.5 Supplementary Analyses

Apart from detailed statistics and learning dynamics of the above results (Appendix K), we further examine performance trends across procedural sources, SOP complexity, backbones, interaction budgets, and experience orders (Appendix N), and report computational cost (Appendix O). These additionally reveal that learning gains remain positive for all evolving methods but do not increase uniformly with backbone capability. Computational cost varies substantially across method categories: methods with the highest FSR gains can require roughly twice the cost of others.

7 Conclusion

We introduce EvoTrace, a benchmark that uses Agent-Executable Standard Operating Procedures as a common reference to track individual procedural requirements across experience, explicit retention, and held-out execution of self-evolving agents. Evaluation of eleven representative methods shows that neither the form nor the coverage of retained knowledge reliably introduces improvements in execution; instead, each mechanism exhibits distinct bottlenecks across learning stages and requirement types, and lightweight stage-targeted interventions confirm these bottlenecks are actionable. These results suggest that robust self-evolution requires mechanism designs that jointly address retention fidelity and execution-time reuse across diverse procedural requirement types.

AI USE STATEMENT

Generative AI tools were used to provide feedback on the research methodology and experimental design, assist with code editing, support the interpretation and presentation of experimental results, and assist with literature search, figure ideation, and language editing. All research decisions, benchmark design choices, implementations, experiments, analyses, and reported claims were reviewed and verified by the authors. Generative AI was not used for auto research. The authors take full responsibility for the final content of the paper and all associated artifacts.

ETHICS STATEMENT

This work studies the evaluation of self-evolving agents on professional procedural tasks. Human experts were involved in reviewing and validating data annotations during benchmark construction and quality control; they were not themselves subjects of the study.

REPRODUCIBILITY STATEMENT

We provide the benchmark resources and implementation in an anonymous repository at https://anonymous.4open.science/r/EvoTrace-20271. The main paper specifies the benchmark formulation, evaluation protocol, and metric; the appendix provides details of dataset construction, runtime and checkpoint protocols, experimental configurations, evidence processing and matching, metric computation, measurement validation, and complete experimental results.

REFERENCES

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. EvoSkill: Automated skill discovery for multi-agent systems, 2026.

Anthropic. System Card: Claude Opus 4.8, May 2026. URL https://www-cdn.anthropic.com/0b4915911bb0d19eca5b5ee635c80fef830a37ea.pdf. Official system card; accessed September 25, 2026.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment, 2025.

Julia Belikova, Rauf Parchiev, Evgeny Egorov, Grigorii Davydenko, Gleb Gusev, Andrey Savchenko, and Maksim Makarenko. Managing procedural memory in LLM agents: Control, adaptation, and evaluation, 2026.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember Me, Refine Me: A dynamic procedural memory framework for experience-driven agent evolution. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16803–16822. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.829.

DeepSeek AI. DeepSeek V4: Technical documentation, April 2026. URL https://fe-static.deepseek.com/chat/transparency/deepseek-V4-model-card-EN.pdf. Official technical documentation; accessed September 25, 2026.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11642–11662. PMLR, 2024.

Eray Can Elumar, Cem Tekin, and Osman Yagan. Cost-aware LLM-based online dataset annotation. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 3516–3559. Curran Associates, Inc., 2025. doi: 10.52202/085713-0125.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.866.

Daocheng Fu, Jianbiao Mei, Rong Wu, Xuemeng Yang, Jia Xu, Ding Wang, Pinlong Cai, Yong Liu, Licheng Wen, and Botian Shi. The agent's first day: Benchmarking learning, exploration, and scheduling in the workplace scenarios, 2026.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*, 2026.

Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. SWE-Skills-Bench: Do agent skills actually help in real-world software engineering?, 2026.

Zhiyuan Jiang, Fangrui Huang, Hanwen Xing, Xander Wu, Yipeng Gao, Rui Cao, Mengdi Wang, Shilong Liu, and Yijiang Li. Demystifying agent skills: Why they work—until they don't, 2026.

Yingtie Lei, Zhongwei Wan, Jiankun Zhang, Samiul Alam, Zixuan Zhong, Peizhou Huang, Xin Wang, Jingxuan Zhang, Donghao Zhou, Yunta Hsieh, Zhihao Dou, Hui Shen, Yan Xu, Dimitrios Dimitriadis, Tuo Zhang, and Mi Zhang. SkillEvolBench: Benchmarking the evolution from episodic experience to procedural skills, 2026.

Zekun Li, Shinda Huang, Jiangtian Wang, Nathan Zhang, Antonis Antoniades, Wenyue Hua, Kaijie Zhu, Sirui Zeng, Chi Wang, William Yang Wang, and Xifeng Yan. SOPBench: Evaluating language agents at following standard operating procedures and constraints, 2025.

Zihan Lin, Zhenyu Chen, Jiawen Wei, Xiaohan Wang, Jie Cao, Jiajun Chai, Wei Lin, Guojun Yin, and Ran He. When not to imitate: Boundary-aware skill memory for reliable tool-use LLM agents, 2026.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12076–12100. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.741.

Subhrangshu Nandi, Arghya Datta, Rohith Nama, Udita Patel, Nikhil Vichare, Indranil Bhattacharya, Prince Grover, Shivam Asija, Giuseppe Carenini, Wei Zhang, Arushi Gupta, Sreyoshi Bhaduri, Jing Xu, Huzefa Raja, Shayan Ray, Aaron Chan, Esther Xu Fei, Gaoyuan Du, Zuhaib Akhtar, Harshita Asnani, Weian Chan, Ming Xiong, Francesco Carbone, and Jeetu Mirchandani. SOP-Bench: Complex industrial SOPs for evaluating LLM agents, 2025.

OpenAI. GPT-5.6 Sol model, 2026a. URL https://developers.openai.com/api/docs/models/gpt-5.6-sol. Official model documentation; accessed September 25, 2026.

OpenAI. GPT-5.6 Terra model, 2026b. URL https://developers.openai.com/api/docs/models/gpt-5.6-terra. Official model documentation; accessed September 25, 2026.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

PayPal. Void an authorized payment, 2026. URL https://developer.paypal.com/checkout/void-authorized-payment/. Updated July 8, 2026; accessed September 25, 2026.

Qwen Team. Qwen3.8-27B, August 2026a. URL https://huggingface.co/Qwen/Qwen3.8-27B. Official model card; accessed September 25, 2026.

Qwen Team. Qwen3.8-Flash-Next: A new architecture, towards ultimate cost-efficiency, August 2026b. URL https://huggingface.co/Qwen/Qwen3.8-Flash-Next. Official model card; accessed September 25, 2026.

Qwen Team. Qwen3.8-2.4T-A95B, August 2026c. URL https://huggingface.co/Qwen/Qwen3.8-2.4T-A95B. Official model card; accessed September 25, 2026.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic Cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.eacl-long.333.

U.S. Environmental Protection Agency. Guidance for preparing standard operating procedures (SOPs). Technical Report EPA/600/B-07/001, U.S. Environmental Protection Agency, April 2007. URL https://nepis.epa.gov/Exe/ZyPURL.cgi?Dockey=P1008GTX.TXT.

Yuyao Wang, Zhongjian Zhang, Mo Chi, Kaichi Yu, Yuhan Li, Miao Peng, Bing Tong, Chen Zhang, Yan Zhou, and Jia Li. EvoMemBench: Benchmarking agent memory from a self-evolving perspective, 2026.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-Memory: Benchmarking LLM agent test-time learning with self-evolving memory, 2025.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 17577–17604. Curran Associates, Inc., 2025. doi: 10.52202/085713-0593.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Y. Zou, and Kunle Olukotun. Agentic Context Engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026a.

Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen Su, Haibo Qiu, Wei Yang, Qingnan Ren, Shun Zou, Wenxuan Huang, Lin Chen, Zehui Chen, and Feng Zhao. SkillFlow: Benchmarking lifelong skill discovery and evolution for autonomous agents, 2026b.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo F. R. Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. SkillLearnBench: Benchmarking continual learning methods for agent skill generation on real-world tasks, 2026.

Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, and Jun Wang. Memento: Fine-tuning LLM agents without fine-tuning LLMs, 2025. Version 2.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, Runyu Yang, Qiangbin Liu, Xinlei Yu, Jianmin Zhou, Na Wang, Chunyang Sun, and Jun Wang. Memento-Skills: Let agents design agents, 2026.

Appendix Contents

A Running Example Overview . . . **14**
B Procedural Sources and Dataset Profile . . . **15**
B.1 Source Materials and Adaptation . . . 15
B.2 Application-Domain Taxonomy . . . 16
B.3 Source Composition and Coverage . . . 16
C AE-SOP Specification, Extraction, and Validation . . . **16**
C.1 Annotation Schema and Granularity . . . 16
C.2 Two-Stage Extraction and Prompt Templates . . . 18
C.3 Reference Validation and Audit . . . 18
D Instance Generation and Split Construction . . . **21**
D.1 Branch-Conditioned Configuration Generation . . . 21
D.2 Factual Consistency and Linguistic Variation . . . 22
D.3 Grouping, Partitioning, and Validation . . . 22
E Runtime and Checkpoint Protocol . . . **22**
E.1 Instance Presentation and Information Boundaries . . . 22
E.2 Instance Disclosure Policy . . . 23
E.3 Persistent State and Checkpoints . . . 23
F Evidence Processing and Matching . . . **24**
F.1 Trace Schema and Operational Units . . . 24
F.2 Experience, Execution, and Artifacts . . . 24
F.3 Canonical Matching and Evidence Decisions . . . 25
F.4 Calibration, Audit, and Reporting . . . 26
G Metric Computation . . . **27**
G.1 Canonical Intersections . . . 27
G.2 Counting and Aggregation . . . 27
G.3 Learning Dynamics . . . 27
H Theoretical Analysis of Aligned Procedural Evidence . . . **28**
H.1 Preliminaries . . . 28
H.2 Aggregate Scores Do Not Identify Execution Profiles . . . 29
H.3 From Requirement Accuracy to Workflow Success . . . 30
H.4 Separating Execution Changes from Retention Changes . . . 32
I Worked Example . . . **33**
I.1 AE-SOP Reference Extraction . . . 33
I.2 Instance Variations . . . 33
I.3 Evidence Computation . . . 34
I.4 Evidence-to-Metric Example . . . 34
J Experimental Setup and Reproducibility . . . **35**
J.1 Evaluation Set and Configurations . . . 35
J.2 Methods, Baselines, and Adaptations . . . 36
K Detailed Metric Statistics . . . **37**
K.1 Execution Results at Checkpoints . . . 37
K.2 Learning Curves across Metrics . . . 38
K.3 Quantitative Learning Dynamics . . . 43
K.4 Numerical Supplement to Main-Text Diagnostic Profiles . . . 45
K.5 Tabulated Artifact Metrics . . . 45
L Extended Evidence-Based Diagnostic Analysis . . . **46**
L.1 Process–Outcome Associations . . . 46
L.2 Backbone Sensitivity of Artifact Construction and Use . . . 46
L.3 Reference-State Composition and Realization . . . 47
L.4 Failure Burden and Requirement-Normalized Violations . . . 48
M From Diagnosis to Targeted Interventions . . . **48**
M.1 Stage-Targeted Interventions versus Original State . . . 48
M.2 Trace-Guided Intervention across Methods . . . 49
M.3 Sensitivity and Limits of Intervention . . . 49
N Robustness and Sensitivity Analyses . . . **51**
N.1 Results by Procedural Source . . . 51
N.2 Performance by Procedure Complexity . . . 51
N.3 Additional Backbone Configurations . . . 51
N.4 Sensitivity across Experience-Budget Strata . . . 52
N.5 Experience-Order Sensitivity . . . 54
N.6 Cross-SOP Experience Mixing . . . 55
O Computational Cost and Resource Efficiency . . . **56**
P Measurement Reliability and Diagnostic Validity . . . **58**

A Running Example Overview

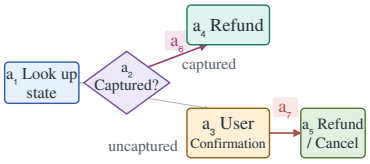
Figure 8 follows a concrete order-refund scenario from source extraction to evaluation and metric computation. Matching colors connect evidence spans to fixed AE-SOP atoms.

Faded elements are unobserved, not retained, or inapplicable; red crosses mark test violations.
Matching colors and IDs link text to graph elements. Rectangles: operations; diamonds: decisions; a_6/a_7: scored dependencies.

One order refund

1 Source → AE-SOP reference

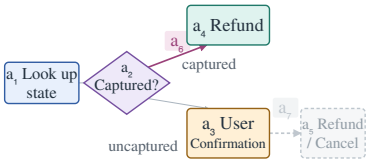
S Look up the order payment state a_1: captured? a_2
→ Captured: refund directly a_4.
→ Uncaptured: request authorization a_3, then refund if yes; cancel if no a_5.
e captured → direct refund a_6; user reply → update a_7.



Reference: all seven atoms

2 Experience

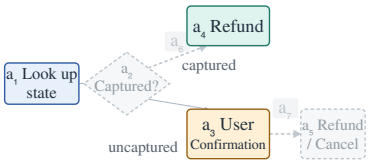
01 User: "Please refund order E101."
02 `get_order_state(E101) → captured` a_1
03 Visible tool feedback: "Captured a_2: refund directly a_4 after this captured-state lookup a_6."
04 Visible policy: "Uncaptured: ask for an authorization reply a_3."
05 `refund(E101) → refunded` a_4



5 observed

3 Artifact

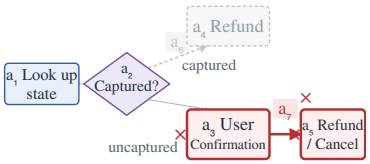
C1 "Look up the order payment state a_1."
C2 "Uncaptured: ask for an authorization reply a_3."
C3 "Refund an order whose payment was captured a_4."
u "Every refund completes within one second." Unmatched.



3 retained

4 Held-out test

01 User: "Please refund order T202."
02 `get_order_state(T202) → uncaptured` a_1
03 Uncaptured a_2 · `refund(T202) → refunded` a_5



2 satisfied · a_3, a_5, a_7 fail (red ×)

Artifact → KP 75% · KR 60% · KC 42.9% Tests → FSR 50% · AMR 70% · K2B 75%

Figure 8: An order-refund walkthrough from AE-SOP extraction to metric computation.

The case separates rules lost during artifact construction from retained requirements omitted in execution. Appendix I gives the full evidence and metric derivation.

B PROCEDURAL SOURCES AND DATASET PROFILE

B.1 SOURCE MATERIALS AND ADAPTATION

Each retained family combines a reusable procedure, executable tools and environment, and a source-native verification anchor. The procedure determines what should be done; the environment makes those operations executable; the verification anchor supplies evidence about their outcomes. Table 3 provides the supplied material and the corresponding annotations by EVOTRACE.

Table 3: Source materials and adaptation.

Source	Supplied source package	AE-SOP adaptation
τ family	*Procedure:* Domain policies. *Execution:* Domain tools and task environments. *Verification:* Source-provided cases and native verification.	Recover conditional rules and user-dependent decisions.
SOPBench	*Procedure:* Service procedures with executable dependencies. *Execution:* Tools for constrained service tasks. *Verification:* Executable procedural constraints.	Align dependency rules with annotated graph elements.
Amazon SOP-Bench	*Procedure:* Industrial SOP documents. *Execution:* Tools and environments for the supplied tasks. *Verification:* Source-native execution verification.	Decompose document instructions into verifiable subgoals.
SkillFlow	*Procedure:* Reusable execution flows. *Execution:* Grounded instances of shared flow structures. *Verification:* Source-provided execution-based grounding.	Preserve topology and add activation and evidence annotations.
AFTER	*Procedure:* Reusable procedural skill documents. *Execution:* Role-specific task execution support. *Verification:* Source-provided execution-based grounding.	Recover task-specific application and composition of skills.
SWE-Skills-Bench	*Procedure:* Skills paired with repository requirements. *Execution:* Repository tasks and execution tools. *Verification:* Acceptance tests.	Map skill requirements to observable repository changes and tests.

Each source is characterized as follows:

The τ benchmark family. Domain policies provide conditional obligations that recur across customer-service instances (Yao et al., 2024). The adaptation separates task-specific entities from policy predicates, grounds decisions in tool observations or user facts, and preserves required confirmations and authorizations when supported by the policy. Policy clauses and native execution checks remain distinct provenance and verification anchors.

SOPBench. Service descriptions paired with executable dependencies provide both procedural claims and constraints on their realization (Li et al., 2025). The adaptation maps independently observable service subgoals to operations and preserves the dependencies enforced by the source. Additional activation and evidence annotations specify when each requirement applies and which trace events establish its satisfaction.

Amazon SOP-Bench. Industrial SOP documents provide longer procedures with conditional steps (Nandi et al., 2025). The adaptation identifies complete procedural goals, separates decisions from operations, and links the resulting elements to document clauses and declared tools. Document order supplies an edge only when the source establishes a data or prerequisite relation.

SkillFlow. Reusable execution flows express shared workflow structure across concrete task instances (Zhang et al., 2026b). The adaptation preserves this structure while grounding operations in the family's tools and specifying the conditions and runtime evidence for each graph element. Different instance entities do not create different procedures when the supported subgoals and dependencies remain the same.

AFTER. Skill documents provide reusable procedural primitives, while task requirements establish how they apply (Belikova et al., 2026). The adaptation combines the relevant skill clauses and task constraints with explicit provenance before extraction. A list of skills is not treated as an ordered workflow; composition edges require support for the corresponding dependency.

SWE-Skills-Bench. Skill documents are grounded in repository requirements and acceptance tests (Han et al., 2026). The adaptation links reusable instructions to observable file, execution, or test outcomes, retaining the repository context needed to run each instance. Passing acceptance tests corroborates the requirements exercised by those tests; it does not by itself establish compliance with every procedural instruction.

B.2 Application-Domain Taxonomy

Each AE-SOP is assigned one mutually exclusive *primary application domain* according to the setting in which its terminal deliverable is used. The classification makes categories recognizable at the task level and prevents double counting by resolving potential overlaps through a fixed hierarchy. A secondary objective is retained separately to describe what the procedure accomplishes within that domain. Source collection, tool type, and file format do not determine the primary label.

Table 4: Application-domain definitions.

Domain	Inclusion criterion
Software Engineering & IT	Software construction, debugging, testing, model engineering, deployment, infrastructure, and observability.
Data, Documents & Analytics	Cross-domain extraction, transformation, reconciliation, statistics, spreadsheets, and document automation.
Finance & Business Analytics	Banking, statements, investment and risk analysis, compensation, sales, and business modeling.
Supply Chain & Industrial Operations	Fulfillment, replenishment, capacity, warehouse operations, dangerous goods, inspection, and recovery planning.
Customer & Public Services	Retail, telecom, travel, hospitality, education, library, motor-vehicle, and general service workflows.
Healthcare	Patient intake, healthcare services, medical-data standardization, and healthcare decision analysis.
Trust, Safety & Compliance	Content safety, fraud and abuse detection, business verification, security review, fact checking, and compliance.

Boundary cases are resolved by the terminal use context. Repository and system-testing workflows remain Software Engineering & IT even when they manipulate data; financial models and banking procedures are Finance & Business Analytics; medical intake and medical-data workflows are Healthcare; and fraud, verification, security, and policy-enforcement workflows are Trust, Safety & Compliance. Industrial inspection, fulfillment, capacity, and recovery tasks map to Supply Chain & Industrial Operations, while customer-facing institutional procedures map to Customer & Public Services. The cross-domain Data, Documents & Analytics label is used only when the deliverable is principally a document, spreadsheet, statistical, or data-transformation artifact without a more specific application setting.

B.3 Source Composition and Coverage

Table 5 presents family counts, experience and test instance counts, from each source. Graph-size summaries count operation nodes, decision nodes, and dependency edges separately, reporting the median and range over retained families.

C AE-SOP Specification, Extraction, and Validation

C.1 Annotation Schema and Granularity

The graph $G_{\mathcal{P}} = (V_O, V_D, E)$ and atoms $a_z = \langle z, \lambda_z, \eta_z \rangle$ follow the main-text specification. Each atom additionally retains source evidence $\rho(z)$. Table 6 details each annotation record.

Operations and Procedural Subgoals. An operation node is

$$o_i = \langle g_i, \pi_i \rangle \in V_O, \tag{4}$$

Source composition and structural depth Darker cells indicate larger values within each column

	Dataset scale			AE-SOP structure: median [range]		
Source	AE-SOPs	Experience	Test	Operations	Decisions	Edges
AFTER	22	1,600	780	14.5 [7-154]	8 [5-63]	27.5 [15-319]
Amazon	11	532	255	11 [4-23]	8 [1-12]	25 [4-44]
SOPBench	7	2,412	1,206	91 [54-113]	73 [45-134]	238 [146-363]
SWE-Skills	48	1,760	780	13 [3-37]	5 [1-17]	21.5 [3-90]
SkillFlow	20	420	189	10.5 [4-22]	4 [0-8]	24 [3-53]
tau	7	576	282	17 [6-47]	12 [5-38]	34 [6-90]

Table 5: Source composition and AE-SOP structural depth. Cell color ranks sources within each column; structural cells report median [range].

Table 6: Annotation record for one graph element.

Field	Content
z	Family-scoped identifier; operation subgoal, decision predicate and outcomes, or dependency endpoints.
λ_z	Task facts and branch conditions under which the requirement applies.
η_z	Observable inputs, outputs, state changes, user feedback, and any ordering, repetition, or termination constraints needed to verify realization.
$\rho(z)$	Source identifier and clause or span supporting the claim; distinct tool declarations and execution evidence where relevant.
Validation	Independent proposals, source-verification decision, execution-corroboration evidence, and expert resolution when applicable.

where g_i is a procedural subgoal and π_i specifies its grounded tool invocations. A subgoal is an independently verifiable procedural outcome: obtaining required information or producing a required state change for the relevant task entities. Its description identifies the entity roles, the required information or effect, and the relations that distinguish the intended outcome from an effect on an unrelated entity. The tool grounding π_i specifies how the declared tools can realize this outcome, with instance-specific values treated as parameters. It is not a transcript that every valid execution must reproduce. Several calls may jointly realize one subgoal, and alternative realizations are accepted when they satisfy the annotated requirements. Internal reasoning and incidental tool calls do not form operation nodes merely because they occur in a trajectory.

Decisions and Conditional Choices. A decision node is

$$d_j = \langle q_j : \mathcal{X}_{\sigma_j} \to \mathcal{Y}_j \rangle \in V_D, \qquad \sigma_j \in \{\texttt{tool}, \texttt{user}\}. \tag{5}$$

Here, $\mathcal{X}_{\sigma_j}$ is the space of observable inputs from source σ_j, $\mathcal{Y}_j$ is the set of execution choices, and q_j maps the relevant input facts to a choice. Tool-sourced facts include observable inputs, outputs, state information, and native errors; user-sourced facts include supplied information, confirmation, and authorization or refusal. The annotation records the input fields and the condition–outcome mapping, not only the names of downstream actions. Decisions with similarly named outcomes remain distinct when their input facts or conditional rules differ. User-intervention points are the user-sourced decisions,

$$V_{\mathcal{P}}^{\texttt{user}} = \{d_j \in V_D : \sigma_j = \texttt{user}\}, \tag{6}$$

which are instantiated by the response specifications in Appendix D.

Dependencies. A directed edge is

$$e_{uv} = \langle u, v \rangle \in E, \qquad u, v \in V_O \cup V_D. \tag{7}$$

It specifies that u supplies data or establishes a prerequisite required by v. The annotations identify the required relation, such as using a previously returned identifier or obtaining authorization before an update. Textual adjacency and an incidental execution order are insufficient support for an edge.

Operational Atoms. Each graph element $z \in V_O \cup V_D \cup E$ defines one operational atom,

$$a_z = \langle z, \lambda_z, \eta_z \rangle, \qquad \mathcal{Z}_{\mathcal{P}}^{\text{ref}} = \{a_z : z \in V_O \cup V_D \cup E\}. \tag{8}$$

The activation condition λ_z determines when the requirement applies; η_z specifies the observable evidence needed to verify its realization. For an operation, this evidence concerns its required information or effect; for a decision, the choice under the relevant observed facts; and for an edge, the required data or prerequisite relation. The specification may include tool inputs and outputs, state changes, user feedback, and ordering, repetition, or termination constraints. A decision rule q_j determines which choice follows from its input, whereas λ_{d_j} determines when the decision is applicable. Failure to establish a mandatory prerequisite does not by itself remove the dependent requirement from the applicable set. Each element also retains source evidence $\rho(z)$ supporting its procedural claim. This provenance is distinct from η_z: the former supports why the requirement belongs in the reference, while the latter specifies how its realization can be checked. Likewise, η_z is an evidence specification, not the particular evidence observed in an execution. Provenance and validation records do not add scoring units.

Applicability and Satisfaction Evidence. For each element, λ_z specifies when the requirement applies, while η_z specifies the observable evidence needed to verify its realization. An operation's evidence rule checks the required information or effect; a decision's rule checks the choice against the relevant facts; and an edge's rule checks the declared data or prerequisite relation. These rules may include entity bindings, tool results, state changes, and ordering, repetition, or termination constraints. The rule η_z is distinct from the evidence observed in a particular trace.

C.2 Two-Stage Extraction and Prompt Templates

The extractor first identifies stable operations and decisions, then reconstructs dependencies and verification annotations with those nodes available. Source text, tool declarations, and source-provided execution grounding are supplied with distinct identifiers. When the extractor is reused for evolved artifacts, the inspected artifact is the source of its claims (Appendix F).

Both extraction stages follow a document-first procedure. The extractor first analyzes the procedural document and constructs source-supported definitions, using the declared tool schemas to establish executable grounding. It then aligns these definitions with corresponding traces to verify their realization under the observed case conditions. Source spans justify the procedural requirements, whereas trace events corroborate their execution and expose unresolved grounding issues. Unexercised requirements remain distinguished from trace-corroborated ones. The following templates exemplify a documented construction procedure.

C.3 Reference Validation and Audit

Three models Claude Opus 4.8 (Anthropic, 2026), GPT-5.6-sol (OpenAI, 2026a), and Qwen-3.8-2.4T-A95B (Qwen Team, 2026c), independently perform extraction. Their proposals are aligned by procedural meaning, including subgoal, condition, and evidence source for nodes and endpoint relation for edges. Equivalent wording counts as agreement; the same action under different conditions does not. A candidate needs equivalent proposals from at least two models. Within a candidate, duplicate formulations are consolidated while preserving their source spans and proposal provenance.

An independent verifier, DeepSeek-V4-Pro (DeepSeek AI, 2026), checks each standardized claim against its cited source. It distinguishes sufficient support, contradiction, and insufficient evidence, escalating unsupported or conflicting interpretations for expert review. Voting agreement alone does not establish source support. Listing 3 specifies the claim-level check.

Source-provided cases are executed and assessed by source-native verifiers. Successful runs corroborate the candidate requirements they exercise, with the supporting trace events recorded for each element. An unexercised element needs explicit source support and remains marked as lacking execution corroboration. Trajectory success does not establish graph completeness or make every observed ordering necessary.

Table 7 separates conditional pipeline checks, cumulative corroboration, and human agreement for references, instances, and scoring.

Listing 1. Stage 1: define operations and decisions from documents, then verify them in corresponding traces.

```
ROLE: Analyze a procedural document, define its operations and decisions, then verify their grounding in
      corresponding execution traces.

DEFINITION INPUT:
FAMILY_ID; DOCUMENT with span IDs; TOOLS with I/O schemas.

VERIFICATION INPUT:
Corresponding TRACES with case facts, trace/event IDs, tool inputs and outputs, user interactions, and
      native-verifier outcomes when available.

PHASE A: DOCUMENT ANALYSIS AND NODE DEFINITION
1. Read DOCUMENT to identify its task scope, procedural subgoals,
   conditional choices, and required outcomes. Separate reusable
   requirements from instance-specific entities and values.
2. Define each operation as o_i = <g_i, pi_i>:
   - g_i specifies an independently observable procedural outcome,
     including the relevant entity roles and required information/effect.
   - pi_i grounds this outcome in the declared tools, using parameter
     roles rather than copying a concrete execution.
   Group calls serving one outcome; separate independently checkable
   subgoals. Internal reasoning is not an operation.
3. Define each decision as d_j = <q_j: X_{sigma_j} -> Y_j>:
   - sigma_j identifies tool- or user-sourced input.
   - X_{sigma_j} specifies the relevant observable facts.
   - q_j specifies how their conditions determine choices in Y_j.
   Preserve the document-supported conditions and alternatives.
4. Cite document spans for every procedural definition and tool-schema
   IDs for its grounding. Record unsupported or ambiguous parts.
   Complete this candidate node inventory before trace verification.

PHASE B: CORRESPONDING-TRACE VERIFICATION
5. Locate traces and events corresponding to each candidate's subgoal,
   entity bindings, and branch conditions.
   - Operation: check whether the observed tool calls and outcomes
     realize g_i; an attempted call is not a successful effect.
   - Decision: identify the observed input and check whether the choice
     agrees with q_j for that input.
6. Record the supporting events, observed outcomes, and discrepancies.
   A trace exercising one branch corroborates that case, not every
   alternative in the decision definition.
   Keep document-supported but unexercised nodes or outcomes explicitly
   marked as lacking trace corroboration.
7. Do not derive obligations from incidental execution patterns or
   silently modify definitions to fit a trace. Request source-grounded
   review when verification reveals an ambiguity or inconsistency.

OUTPUT: JSON {
  family_id, scope,
  operations: [{
    id, subgoal, observable_outcome, tool_grounding,
    source_span_ids, tool_schema_ids
  }],
  decisions: [{
    id, predicate, evidence_source, input_fields,
    outcomes: [{condition, choice}], source_span_ids
  }],
  trace_checks: [{
    element_id, trace_id, event_ids, input_facts, observed_result,
    verdict: corroborated|execution_violation|insufficient,
    explanation
  }],
  uncovered: [{element_id, unexercised_outcomes}],
  unresolved: [{element_or_span_id, issue, evidence_needed}]
}.
Do not construct dependency edges at this stage.
```

Listing 2. Stage 2: define dependencies and atom annotations from documents, then verify them in corresponding traces.

```
ROLE: Analyze document-supported relations between the defined nodes,
construct dependencies and atom annotations, then verify them in traces.

DEFINITION INPUT:
DOCUMENT with span IDs; TOOLS with I/O schemas;
stage-1 NODE_DEFINITIONS and trace-check reports.

VERIFICATION INPUT:
Corresponding TRACES with case facts, trace/event IDs, tool inputs and
outputs, user interactions, and native-verifier outcomes when available.

PHASE A: DOCUMENT ANALYSIS AND ATOM DEFINITION
1. Re-read DOCUMENT using the stage-1 nodes as anchors. Identify where
   one node supplies data or establishes a prerequisite required by
   another. Define each dependency as e_uv = <u, v>, recording the
   required relation and its supporting document spans.
   Neither textual adjacency nor observed execution order alone
   establishes a necessary dependency.
2. For every operation, decision, and dependency z, define
   a_z = <z, lambda_z, eta_z>:
   - lambda_z specifies the document-supported applicability conditions.
   - eta_z specifies the observable evidence required for realization:
     inputs, outputs, state changes, user responses, or event relations.
   Use the tool schemas to make these evidence requirements checkable.
3. Preserve source-supported ordering, repetition, and termination
   constraints. Reuse node IDs and create one atom per graph element.
   Keep source provenance rho(z) separate from the evidence rule eta_z.
   Complete the candidate annotations before inspecting their
   realization in the corresponding traces.

PHASE B: CORRESPONDING-TRACE VERIFICATION
4. For each atom, identify the relevant trace and bind its entity roles
   and conditions to the recorded case facts.
   Determine applicability before checking realization; failure to
   establish a mandatory prerequisite does not make its dependent
   requirement inapplicable.
5. Check the observed events against eta_z:
   - Operation: verify the required information or effect.
   - Decision: verify the choice under the applicable observed facts.
   - Dependency: verify the required data binding or prerequisite
     relation, not merely the presence of both endpoints.
   Inspect the relevant event sequence for ordering, repetition,
   and termination constraints.
6. Record concrete supporting or violating events and missing evidence.
   Separate an execution violation from an unsupported definition:
   a failed trace does not by itself invalidate a document requirement,
   and native task success does not verify every procedural constraint.
7. Flag unexercised requirements and source--trace discrepancies.
   Any definition change requires renewed document support and
   re-verification; do not silently repair annotations from a trace.

OUTPUT: JSON {
  edges: [{
    id, from, to, dependency, source_span_ids
  }],
  atoms: [{
    element_id, element_type, activation_conditions,
    satisfaction_evidence, source_span_ids
  }],
  trace_checks: [{
    element_id, trace_id, applicability, event_ids, observed_evidence,
    verdict: corroborated|execution_violation|inapplicable|insufficient,
    explanation
  }],
  uncovered: [{element_id, unexercised_conditions}],
  unresolved: [{element_ids, issue, evidence_needed}]
}.
```

Listing 3. Independent claim–source verification.

```
ROLE: Independently verify a procedural claim and its evidence rule.
INPUT: CLAIM with conditions; EVIDENCE_RULE; cited SOURCE spans and context; relevant TOOLS.
Check source support for the complete claim, including conditions, effects, and dependencies. For edges,
verify necessity rather than textual order. Check that the evidence rule distinguishes satisfaction from
attempts, wrong branches, and unmet prerequisites. Neither extractor agreement nor outside rules supply
evidence.
OUTPUT: JSON {verdict: supported|contradicted|insufficient, supporting_span_ids,
  conflicting_span_ids, unsupported_claim_parts, evidence_rule_issues, explanation,
  requires_expert_review}.
Escalate contradictions, insufficient support, or unresolved evidence-rule issues. Give a brief
evidence-based explanation; a supported claim does not establish graph completeness.
```

Table 7: Construction-validation rates.

Measure	Counting unit or denominator	Value
Extraction agreement	Aligned candidates with agreed LLM proposals	89.0%
Claim–evidence support	Agreed candidates supported by source evidence	90.3%
Trajectory corroboration	Source-supported candidates exercised in traces	88.2%
End-to-end corroboration	Trace-corroborated candidates among all candidates	70.8%
Expert-review agreement	Observed pre-adjudication pairwise expert agreement	82.6%
Human instance agreement	Instance validity: 240 cases, 720 pairs, 642 agreeing	89.2%
Human scoring agreement	Scoring credit: 240 items, 720 pairs, 578 agreeing	80.3%
Reference completeness	Requirements recovered against the expert reference inventory	86.9%

Extraction: 3 LLMs, $\geq$2 agreeing. Human agreement: 3 experts, pairwise before adjudication.
Reference review: 596 items, 440 unanimous; 1,476 agreeing pairs among 1,788 pairs.

Pairwise agreement is 82.6% for reference review (1,476 agreeing pairs among 1,788), 89.2% for instance validity (642 agreeing pairs among 720), and 80.3% for scoring credit (578 agreeing pairs among 720). The conditional checks yield 70.8% end-to-end corroboration, so a high pass rate at an individual stage does not imply complete references. This gap arises because each stage filters a subset of the input: extraction agreement eliminates unsupported candidates, claim-evidence support removes contradicted claims, and trajectory corroboration excludes unexercised requirements. Reviewers assess source consistency and missing requirements independently before adjudicating unsupported elements, omissions, conditions, and dependencies.

D Instance Generation and Split Construction

D.1 Branch-Conditioned Configuration Generation

For each family, the generation input comprises its annotated graph, tools and environment, source-provided runnable cases where available, and target requirements or activation patterns. Starting with coverage of the source cases, the generator selects under-covered requirements and traces backward through their conditions and prerequisite dependencies. This identifies a jointly consistent combination of initial state, earlier operation outcomes, and user-provided facts. Independent edits to entities or branch facts are insufficient when they invalidate a downstream precondition.

A concrete configuration supplies m_x^0 and $\mathcal{R}_x$ together with the environment initialization needed to realize their facts. The configuration record includes entity identifiers, relevant attribute values and files, initial state, user-dependent interaction points, required response content, and expected applicable atoms. The request communicates the task; the response specification retains later user facts until their allowed interaction points. Parameter values may vary while the procedure and targeted activation pattern remain fixed.

Each candidate is executed in the family environment before language expansion. The check establishes runnable initialization and tool interactions, assesses the source-native verifier, and compares realized activation with the expected applicable requirements. Conflicting facts, failed execution, or incorrect branch activation lead to revision and rechecking or exclusion. The accepted configuration retains its validation trace, verifier outcome, and realized activation record.

Listing 4. Generate a branch-conditioned instance configuration.

```
ROLE: Construct an executable instance of a fixed SOP.
INPUT: GRAPH_AND_ANNOTATIONS; TARGET requirements/activation pattern; ENVIRONMENT schema and
constraints; runnable SEEDS if available; allowed ENTITY_POOL.
1. Work backward from TARGET through decisions and dependencies to determine required outcomes,
     environment conditions, and user facts.
2. Instantiate compatible entities, values, files, and state; propagate dependent changes while
     preserving the procedure and tool semantics.
3. Write the initial request and later response specifications. Keep later facts at their designated
     disclosure points; reveal no hidden graph, scoring annotations, or solution.
4. Derive applicable atoms, flag incompatible conditions or unsupported initialization, and propose
     execution/activation checks. Do not claim that these checks were run.
OUTPUT: JSON {status: candidate|infeasible, configuration_id, target_requirements,
  prerequisite_conditions, entities, attributes, environment_initialization, initial_request,
  responses: [{interaction_id, fixed_facts, required_content}], expected_applicable_atoms,
  validation_checks, unresolved_constraints}.
```

D.2 Factual Consistency and Linguistic Variation

The facts of a validated configuration are frozen before generating language variants. A linguistic template specifies the request and response expression pattern; paraphrases vary wording within that pattern. Variation may change expression style and information order where permitted, but preserves entities, attribute values, intent, required response content, and disclosure points. Each realization is checked against the frozen configuration rather than against another generated paraphrase.

Listing 5. Generate and check fact-preserving language variants.

```
ROLE: Generate or check a fact-preserving linguistic realization.
INPUT: MODE=generate|check; frozen CONFIGURATION (m0, responses); TEMPLATE and template_id; CANDIDATE
when checking.
GENERATE: Render the request and responses using TEMPLATE. Vary wording, tone, and permitted information
order; preserve entities, values, conditions, intent, response facts, and consent scope. Add no
procedural instruction or prematurely disclosed fact.
CHECK: Compare directly with CONFIGURATION for omitted, altered, or added facts, changed consent,
branch-changing implications, and premature disclosure. Flag failures rather than silently repairing
them.
OUTPUT: JSON {mode, configuration_id, template_id, initial_request, responses,
  checks: [{field_or_interaction, status: pass|fail, evidence}],
  verdict: candidate|consistent|inconsistent}.
Generation returns candidate; only check mode can return consistent. Revise and recheck failures.
```

D.3 Grouping, Partitioning, and Validation

Group variants share a configuration, entity identifier, or template before allocation; conflicting groups are resolved by regenerating configurations or templates. Experience and test thus share the governing procedure while differing in concrete configurations and expression templates. After allocation, compare the configuration identifiers, entity identifiers, and template identifiers across splits and recheck fact consistency for retained realizations. Shared attribute categories or branch outcomes are permitted: their overlap represents reuse of the same procedure. Constructed activation is not substituted for evidence an evaluated agent actually observed.

E Runtime and Checkpoint Protocol

E.1 Instance Presentation and Information Boundaries

An instance $I_x^{\mathcal{P}} = \langle m_x^0, \mathcal{R}_x \rangle$ separates information available at task entry from information supplied through subsequent user interaction. The initial request and task attachments are delivered through m_x^0. For a user-dependent interaction point v, $\mathcal{R}_x(v)$ specifies fixed user facts and required response content; it does not grant permission to reveal those facts before the associated disclosure conditions hold. Concrete wording may vary only within the validated realization of this specification. `Agent Runner` creates a fresh task session, initializes the instance environment, and registers the family's declared tools. Tools return their actual outputs and native errors; reference graphs, requirement

annotations, and expected solutions are withheld from the agent. The `Adaptive User Simulator` provides task input and conditional in-process responses. It delivers the task description and task-related attachments through m_x^0, and conditionally releases user responses $\mathcal{R}_x(v)$ according to the disclosure policy Φ_x.

E.2 Instance Disclosure Policy

Φ_x maps user-observable execution states to conditional response disclosure decisions. It first translates the activation condition λ_v and the runtime evidence η_v of the user-intervention node $v \in V_{\mathcal{P}}^{\text{user}}$ into runtime rules. Specifically, λ_v is realized as a disclosure guard g_v that judges whether the current user-observable state (including tool calls, inputs/outputs, and environment states) satisfies the preconditions for releasing the response; η_v is realized as an expected user-observable event e_v that characterizes the agent's user-facing inquiry corresponding to this node. At runtime, Φ_x monitors the agent execution, and releases response $\mathcal{R}_x(v)$ only when both the expected event e_v matches and the guard g_v is satisfied. Here, event matching is performed through deterministic matching when the event has clear structural patterns (e.g., tool calls, specific values), and through semantic matching when it describes a concept.

The following listings are templates implementing this specification. Placeholders denote role-specific inputs.

Listing 6. Disclosure event and guard decision template.

```
ROLE: Decide whether a specified user response may be released.
INPUT: INSTANCE_ID; current user-observable event and VISIBLE_HISTORY; CANDIDATE_RULES {interaction_id,
expected_event, guard, authorized_payload_id, replay_rule}; DETERMINISTIC_CHECKS with event IDs;
OFF_PATH_POLICY; PRIOR_RELEASE_LOG.
1. Use only visible evidence, never private reasoning, hidden state, solutions, scoring annotations, or
  agent artifacts. Match the inquiry's goal, entity, action, and scope; accept valid paraphrases.
2. Cite evidence for each guard and respect deterministic failures. Missing evidence does not establish
  a true guard.
3. Release only if event matching, the guard, and the replay rule all permit it. For ambiguous/unmatched
  requests, apply OFF_PATH_POLICY without changing facts or inventing payloads.
OUTPUT per rule: {interaction_id, event_match: yes|no|insufficient,
  guard: true|false|insufficient, visible_evidence_ids, release: yes|no,
  authorized_payload_id|null, off_path_action|null, brief_justification}.
Return controller records, not a user message.
```

Listing 7. Authorized user-response rendering template.

```
ROLE: Render one user message from authorized content.
INPUT: current inquiry and permitted context; RELEASE_DECISION; APPROVED_PAYLOAD (facts, required
content, linguistic realization, consent/refusal); APPROVED_OFF_PATH_RESPONSE if supplied.
1. On release, answer concisely using only APPROVED_PAYLOAD. Preserve facts, IDs, amounts, and consent;
  add no new facts, instructions, or suggestions for improving the agent.
2. Reveal no later facts, guards, reference steps, or scores. Clarification is not consent; never
  reverse refusal or repair the agent's chosen branch.
3. On denied release, use only APPROVED_OFF_PATH_RESPONSE. If none exists, return NO_AUTHORIZED_RESPONSE
  to the controller rather than inventing a message.
OUTPUT: {response_text|NO_AUTHORIZED_RESPONSE, payload_id, included_fact_ids}.
Deliver only response_text to the evaluated agent.
```

E.3 Persistent State and Checkpoints

Let S_t denote the complete persistent state at checkpoint c_t; A_t is its inspectable artifact view when available. The adapter's state inventory enumerates mutable memories, prompt and skill files, persistent stores, and other method-specific components, together with their snapshot and restore procedures. Saving an artifact while leaving an untracked memory store shared across tests is not a complete snapshot, because the shared store can accumulate cross-test contamination or retain information from earlier held-out instances. This incomplete isolation would invalidate the checkpoint comparison by allowing evolution during testing. Session context, instance environment state, and task-local scratch storage are reset for every instance.

At each checkpoint c_t, the evidence package C_t contains $\langle T_t^{\text{train}}, A_t, T_t^{\text{test}} \rangle$, where T_t^{train} is the cumulative training trajectories produced up to evolution checkpoint c_t; A_t is the evolved artifact snapshot on these trajectories; and T_t^{test} is the test trajectories produced from a frozen copy of A_t on held-out test set instances.

The control protocol of the `Agent Runner` is as follows:

Listing 8. Checkpoint isolation and attempt accounting.

```
S := declared initial persistent state; n := 0
for x in the fixed experience order:
    reset session and environment; deliver request; register attempt_id
    if delivery failed: follow infrastructure policy
    else:
        n := n + 1; run under resource limits
        log visible events, failures, and permitted evolution updates
        S := complete state after the permitted update
    if n is a checkpoint boundary and this attempt is finalized:
        C := immutable snapshot of S
        for x_test in the fixed held-out set:
            reset session, environment, and task-local storage
            S_test := independent copy of C; disable evolution/feedback
            run x_test; retain evaluator logs; discard S_test/local state
        score without writing to C or S; verify integrity; resume
Infrastructure retries reuse the original attempt_id; do not recount.
```

F Evidence Processing and Matching

The evidence package at c_t contains cumulative acquisition traces T_t^{train}, an inspectable snapshot A_t where available, and frozen test traces T_t^{test}. Extraction records what each source supports before matching it to a procedural requirement. Reference annotations guide evaluation; they cannot supply missing observations to an acquisition trace or missing instructions to an artifact.

F.1 Trace Schema and Operational Units

A tool execution is recorded as

$$\pi_\ell = \langle e_k, f_\ell, p_\ell, r_\ell, \Delta s_\ell \rangle,$$

where e_k references its observable triggering context, f_ℓ is the tool identity, p_ℓ and r_ℓ are inputs and returned outputs or native errors, and Δs_ℓ records relevant observable state changes. Each record includes its event order, instance and family identifiers, and visibility flags. Trigger references may identify a user message, a tool result, or a preceding public action; they do not claim access to the agent's internal reason for acting. An unavailable state delta is marked unavailable rather than reconstructed from an expected outcome.

Executions are segmented into event-coherent subsequences sharing the same supported triggering context. Each segment yields an operational unit $\langle \widehat{g}_k, \widehat{\eta}_k \rangle$: a procedural goal supported by the observable context and the corresponding tool, result, and state-change evidence. Segmentation preserves event identifiers and ordering.

F.2 Experience, Execution, and Artifacts

Acquisition-Observed Atoms. $\widehat{\mathcal{Z}}_t^{\text{train}}$ is extracted from evidence actually available to the agent across attempted experience instances. Informative tool errors and user corrections may expose a condition even when the agent fails to execute it correctly. Conversely, reaching a requirement in a reference trace, attempting an operation, or seeing two actions in sequence does not alone establish observation of the corresponding conditional rule or necessary dependency. Each extracted atom cites the visible evidence supporting its content; partial exposure is retained as partial evidence rather than completed from hidden annotations.

Test-Execution Atoms. $\widehat{\mathcal{Z}}_{t,x}^{\text{test}}$ records requirements supported by execution evidence for test instance x. Operations require the declared effects, decisions require behavior consistent with the applicable facts, and dependencies require the annotated data flow or prerequisite relation. A manda-

Listing 9. Trajectory-to-evidence mapping template.

```
ROLE: Extract operational units and auditable evidence.
INPUT: MODE=acquisition_observation|test_execution; family/instance IDs; ordered events with visibility
flags; tool schemas, I/O and state deltas; annotations for later alignment.
1. Index observable events and group calls by supported triggering context. Preserve event order, entity
     bindings, errors, and non-tool user interactions; invent no events or private reasoning.
2. Describe each unit's supported goal separately from its tool evidence. Mark missing observations
     explicitly.
3. Acquisition: use only evidence delivered to the agent; identify exposed content rather than planned
     coverage. Test: record satisfaction/violation evidence, including prerequisites and ordering;
     claimed completion is not a tool effect.
4. Keep observations separate from reference expectations. Mark partial, contradictory, or absent
     support; do not complete traces or compute aggregate metrics.
OUTPUT: {units: [{unit_id, triggering_event_ids, supported_goal, tool_event_ids, entity_bindings,
     observed_effects}], claims: [{claim_id, content, conditions, dependencies, evidence_ids,
     visibility, mode, candidate_requirement_ids, support: complete|partial|contradictory|absent}],
     unresolved_cases_with_explanations}.
```

tory downstream requirement remains applicable after an upstream agent failure. Unreached or unsupported mandatory requirements do not become successes or disappear from the denominator. Inapplicable alternatives are distinguished from failed applicable requirements.

Explicit Artifact Atoms. The `AE-SOP Extractor` is reused on the actual inspectable contents of A_t, with artifact locations replacing source-document provenance. It extracts claims before reference matching and preserves applicability, decision conditions, dependencies, and explicit prohibitions. Text and executable code can encode procedural content, but a code path's presence does not establish that it was executed or is operationally reachable. Omitted conditions are not restored from an oracle reference. Unmatched and contradicted claims remain in the extracted inventory; an unavailable artifact view is not represented as an empty artifact.

F.3 Canonical Matching and Evidence Decisions

Matching assigns evidence-supported claims to canonical atoms $a_z = \langle z, \lambda_z, \eta_z \rangle$. For execution traces, we use the operational units $\langle \widehat{g}_k, \widehat{\eta}_k \rangle$ together with the original event stream; for artifacts, use the extracted claims and their text or code locations.

Type-Specific Comparison. For *operation* $o_i = \langle g_i, \pi_i \rangle$, compare the procedural goal g_i with the trajectory-supported $\widehat{g}_k$ or the artifact's extracted subgoal. The semantic comparison checks the required information or effect, entity roles, and applicability expressed by the claim. For execution, deterministic checks apply the executable parts of η_{o_i} to the recorded $(f_\ell, p_\ell, \Delta s_\ell)$: permitted tool realization, argument bindings, returned results, and required state effects. In the running example, `Refund` requires a matching refund result. `Refund / Cancel` additionally requires agreement with the user's confirmation reply: approval permits refund, whereas refusal requires cancelling the authorization. A call with the correct name but an error result is insufficient for execution satisfaction.

For *decision* $d_j = \langle q_j : \mathcal{X}_{\sigma_j} \to \mathcal{Y}_j \rangle$, compare the input source σ_j, relevant input fields, and condition–outcome mapping q_j. In execution, recover the applicable input $\xi_j \in \mathcal{X}_{\sigma_j}$ and selected choice $\widehat{y}_j \in \mathcal{Y}_j$ from the annotated evidence. When the input and rule are executable, the deterministic comparison is $\widehat{y}_j = q_j(\xi_j)$. For natural-language input, semantic interpretation identifies its factual content and scope: for example, whether a reply authorizes the specified update, refuses it, or only requests clarification. If the rule itself is non-executable, a semantic check compares the cited facts and chosen outcome under the annotated rule; it cannot invent missing inputs.

For *dependency* $e_{uv} = \langle u, v \rangle$, first align the contents of the endpoints, then compare the necessary data or prerequisite relation. In experience or artifact text, semantic checks distinguish an explicit requirement such as "for uncaptured payments, obtain the user's confirmation before refunding or cancelling the authorization" from an incidental list of actions. For execution, deterministic checks relate the corresponding concrete events e^u, e^v through their original positions and bound data.

Joint Evidence Decision. All checks use one consistent alignment of claims, events, and entity roles. Let $D_r(B, a_z; b)$ and $S_r(B, a_z; b)$ be the aggregate deterministic and semantic check results under b, respectively, each in $\{1, 0, -\}$. Within each group, 0 means an established contradiction to at least one required check; $-$ means at least one required check is unresolved and none is con-

tradicted; 1 means all required checks are supported. A group with no required checks returns 1, but the complete checklist always requires identifiable supporting evidence. A missing observation never becomes an empty checklist.

The decision for this alignment is

$$M_r(B, a_z; b) = \begin{cases} \texttt{supported}, & D_r(B, a_z; b) = S_r(B, a_z; b) = 1, \\ \texttt{contradicted}, & D_r(B, a_z; b) = 0 \text{ or } S_r(B, a_z; b) = 0, \\ \texttt{insufficient}, & \text{otherwise.} \end{cases} \tag{9}$$

For alternative valid realizations, test each admissible alignment with both check groups together. An atom is accepted only through a jointly passing alignment, including all applicable whole-trace constraints. A failed speculative alignment is not itself a violation of the complete execution. On a complete trace, absence of a required event fails its existence check; an incomplete log leaves that check unresolved.

Each decision records the relation, canonical atom ID, evidence locations, bindings, per-check results, checker or rubric version, and unresolved fields. Semantic judgments cite the evidence for the relevant goal, condition, effect, or relation.

Listing 10. Evidence-grounded semantic requirement matching template.

```
ROLE: Judge a specified evidence relation rather than word overlap.
INPUT: RELATION=observed_in_experience | encoded_in_artifact | satisfied_in_execution |
artifact_supported_by_experience;
LEFT_CLAIM and RIGHT_CLAIM_OR_REQUIREMENT; family/entity scope; cited evidence, applicability/evidence
specifications, deterministic results, canonical-ID candidates.
1. Compare goals, conditions, alternatives, effects, entity relations, and dependencies. Accept
  supported equivalents; infer no missing facts, conditions, or tool effects.
2. Respect deterministic failures: semantic similarity cannot override them. Observation, encoding, and
  execution require their own relation-specific evidence.
3. For artifact_supported_by_experience, use actual visible acquisition evidence; require reference
  membership only when requested.
4. Preserve family scope and deduplicate genuine equivalents, not unrelated claims sharing a tool or
  verb. Do not present missing procedure text as observed.
OUTPUT: {decision: supported|contradicted|insufficient, matched_canonical_id|null,
  evidence_ids, condition_check, dependency_check, effect_check, deterministic_constraints,
  brief_rationale, ambiguity_or_adjudication_flag}.
```

F.4 Calibration, Audit, and Reporting

Table 8 summarizes binary automatic-versus-reference agreement by judgment channel, using the same counts as Table 19.

Table 8: Channel-level agreement rates; false acceptance/rejection use reference-negative/positive denominators.

Judgment channel	Sample size	Agreement	False accept	False reject
Exposure support	240	87.5%	13.0%	12.2%
Artifact matching	240	82.1%	22.6%	16.3%
Operation execution	80	88.8%	15.0%	10.0%
Decision execution	80	78.8%	23.1%	20.4%
Dependency execution	80	76.3%	34.5%	17.6%
Complete workflow	120	84.2%	15.8%	15.9%

Decision and dependency judgments agree less often than operation judgments (78.8% and 76.3% versus 88.8%), reflecting the greater difficulty of verifying conditional rules and prerequisite relations compared to checking tool effects. The complete-execution rate of 84.2% compares binary all-requirements outcomes, aggregating satisfaction across all element types. Only `supported` receives credit; `contradicted` and `insufficient` do not.

G METRIC COMPUTATION

G.1 CANONICAL INTERSECTIONS

An atom receives credit when at least one admissible alignment passes both check groups, including every applicable whole-scope constraint. The main-text intersections are therefore

$$\begin{aligned}\widehat{\mathcal{Z}}_t^{\text{train}} \cap \mathcal{Z}^{\text{ref}} &= \{a_z \in \mathcal{Z}^{\text{ref}} : \exists b :\ M_{\text{obs}}(T_t^{\text{train}}, a_z; b) = \texttt{supported}\},\\ \widehat{\mathcal{Z}}_t^{A} \cap \mathcal{Z}^{\text{ref}} &= \{a_z \in \mathcal{Z}^{\text{ref}} : \exists b :\ M_{\text{enc}}(A_t, a_z; b) = \texttt{supported}\},\\ \widehat{\mathcal{Z}}_{t,x}^{\text{test}} \cap \mathcal{Z}_x^{\text{ref}} &= \{a_z \in \mathcal{Z}_x^{\text{ref}} : \exists b :\ M_{\text{exe}}(\tau_{t,x}, a_z; b) = \texttt{supported}\}.\end{aligned} \tag{10}$$

G.2 COUNTING AND AGGREGATION

For the selected SOP families $\mathcal{F}$, the complete reference is

$$\mathcal{Z}^{\text{ref}} = \bigcup_{\mathcal{P}\in\mathcal{F}} \mathcal{Z}_{\mathcal{P}}^{\text{ref}}. \tag{11}$$

Reference identifiers include the family and graph-element identifiers. Thus, similarly worded requirements in distinct families remain distinct. Each operation, decision, or dependency contributes one atom. Repeated observations of the same atom are deduplicated within a cumulative acquisition set; repeated realization within a test instance earns no additional credit. Repetition constraints, when present, are checked as part of the annotation.

For instance x, $\mathcal{Z}_x^{\text{ref}}$ is determined by the instance facts and the reference activation conditions. Omitting a mandatory prerequisite does not remove its dependent requirement from the scoring denominator. Conversely, an operation belonging exclusively to an inapplicable branch is not required. Conditions, prerequisites, and alternative realizations follow the annotation rules in Appendix C.

For concise computation, let

$$\begin{aligned}\mathcal{E}_t &= \widehat{\mathcal{Z}}_t^{\text{train}} \cap \mathcal{Z}^{\text{ref}},\\ \mathcal{R}_t &= \widehat{\mathcal{Z}}_t^{A} \cap \mathcal{Z}^{\text{ref}},\\ \mathcal{C}_{t,x} &= \widehat{\mathcal{Z}}_{t,x}^{\text{test}} \cap \mathcal{Z}_x^{\text{ref}}.\end{aligned} \tag{12}$$

These sets contain, respectively, observed reference requirements, explicitly retained reference requirements, and correctly realized applicable test requirements. An artifact's distinct unmatched claims remain in $\widehat{\mathcal{Z}}_t^{A}$ and the KP denominator but receive no KP credit. KR counts acquisition–artifact overlap within the reference through $\mathcal{R}_t \cap \mathcal{E}_t$.

G.3 LEARNING DYNAMICS

Let M_t be one of the checkpoint metrics and n_t the cumulative number of attempted experience instances at checkpoint c_t. The same held-out test set is used at every checkpoint. Failed agent attempts contribute to n_t under the accounting rule in Appendix E. The definitions below apply to each recorded run; the selected summaries, checkpoint schedule, and any thresholds are fixed in the run configuration before comparing methods. The present evaluation uses the selections specified above; the definitions support additional analyses under the same framework.

Learning Gain and Acquisition Speed. Improvement relative to the initial checkpoint is

$$\Delta M_t = M_t - M_0. \tag{13}$$

The observed acquisition speed between adjacent checkpoints is

$$s_t(M) = \frac{M_t - M_{t-1}}{n_t - n_{t-1}}, \qquad n_t > n_{t-1}. \tag{14}$$

This is a finite-difference rate per attempted experience instance, not a derivative of an observed continuous process. It accounts for unequal checkpoint spacing. Negative values indicate a decrease in the measured score over that interval. Comparisons use the same metric, experience unit, and acquisition schedule.

Sample Efficiency. For a fixed experience budget, the normalized baseline-adjusted area under the learning curve is

$$\mathrm{AULC}(M) = \frac{1}{n_T - n_0} \sum_{t=1}^{T} (n_t - n_{t-1}) \left(\frac{M_t + M_{t-1}}{2} - M_0 \right), \qquad n_T > n_0. \tag{15}$$

A complementary threshold summary is the first recorded additional experience count $n_t - n_0$ at which $M_t \geq q$, for a threshold q fixed in advance. If the initial checkpoint meets the threshold, the required additional experience is zero. If the threshold is never reached, the result is right-censored at the final budget; it is not assigned an artificial hitting time. This summary reports observed checkpoint attainment, without inferring an unobserved crossing between checkpoints.

Plateaus and Stability. The plateau specifies a tolerance $\epsilon \geq 0$ in score units per experience instance and a window of $w \geq 1$ consecutive intervals. A plateau is confirmed at the first checkpoint whose preceding w interval rates all satisfy $|s_t(M)| \leq \epsilon$. The confirmation point and the covered experience interval are reported together. Subsequent changes outside this tolerance are visible in the full curve and invalidate any claim of a sustained plateau over those later intervals.

Retention Under Intervening Experience. A retention analysis requires a specified acquisition order and revisits to the same earlier test set. For family $\mathcal{P}$, let $t_{\mathcal{P}}$ denote the checkpoint immediately after its designated acquisition block and T a later checkpoint after the declared intervening experience. The retention change is

$$\mathrm{RetChange}_{\mathcal{P}} = M_{T,\mathcal{P}} - M_{t_{\mathcal{P}},\mathcal{P}}. \tag{16}$$

A negative value indicates lower performance on that earlier family's tests. Both measurements use frozen copies of the corresponding state, and testing does not itself supply subsequent acquisition experience. This analysis is only applicable when the acquisition order and revisit schedule are recorded; an ordinary learning-curve fluctuation alone is not evidence of interference-induced forgetting.

H Theoretical Analysis of Aligned Procedural Evidence

EvoTrace connects experience, retained artifacts, and execution through shared requirement identities. The following results explain how this representation distinguishes execution profiles hidden by aggregate scores, relates requirement accuracy to workflow success, and separates changes in execution from changes in retention across checkpoints.

H.1 Preliminaries

Fix a run, a checkpoint, and a held-out set $\mathcal{X}$ of $N > 0$ instances. Let $\mathcal{Z}$ denote the reference universe, with $M = |\mathcal{Z}| > 0$, and let $\mathcal{Z}_x \subseteq \mathcal{Z}$ be the nonempty set of requirements applicable to instance x. Each requirement is identified by its SOP family and graph element, as in Appendix G. Applicability is determined by instance facts and reference conditions; a mandatory step remains in $\mathcal{Z}_x$ even if the agent fails to reach it.

After evidence matching, write

$$\begin{aligned} O &= \widehat{\mathcal{Z}}^{\mathrm{train}} \cap \mathcal{Z}, \qquad R = \widehat{\mathcal{Z}}^{A} \cap \mathcal{Z}, \\ C_x &= \widehat{\mathcal{Z}}_x^{\mathrm{test}} \cap \mathcal{Z}_x, \quad X_a = \{x \in \mathcal{X} : a \in \mathcal{Z}_x\}. \end{aligned} \tag{17}$$

Here O contains requirements observed in experience, R contains those retained in the artifact, and C_x contains those successfully executed on instance x. The set X_a collects the tests on which requirement a applies.

For each requirement $a \in \mathcal{Z}_+ = \{a : |X_a| > 0\}$ with at least one applicable test, define its execution success rate and the eligible retained set as

$$r(a) = \frac{|\{x \in X_a : a \in C_x\}|}{|X_a|}, \qquad R_+ = R \cap \mathcal{Z}_+. \tag{18}$$

For $R_+ \neq \emptyset$, define an auxiliary mean over retained requirements:

$$\bar{r}_R = \frac{1}{|R_+|} \sum_{a \in R_+} r(a). \tag{19}$$

This requirement-equal mean is distinct from the reported trace-equal K2B. With $U_R = \{x : |R \cap \mathcal{Z}_x| > 0\}$, the benchmark uses

$$\text{K2B} = \frac{1}{|U_R|} \sum_{x \in U_R} \frac{|R \cap C_x|}{|R \cap \mathcal{Z}_x|}. \tag{20}$$

The set $B = \bigcup_x C_x = \{a \in \mathcal{Z}_+ : r(a) > 0\}$ records which requirements succeed at least once, while $r(a)$ records how often they succeed.

To describe the diagnostic profiles in Appendix L, let

$$\begin{aligned} G_{ij} &= \{a \in \mathcal{Z}_+ : \mathbf{1}\{a \in O\} = i,\ \mathbf{1}\{a \in R\} = j\}, \\ n_{ij} &= |G_{ij}|, \qquad \mu_{ij} = \frac{1}{n_{ij}} \sum_{a \in G_{ij}} r(a) \quad (n_{ij} > 0). \end{aligned} \tag{21}$$

Thus μ_{11} is the mean success rate for observed and retained requirements, and μ_{01} is the mean for retained requirements without recorded exposure. These means define the group colors in the diagnostic profiles. When pooling runs, requirement identities also include the run identifier.

H.2 Aggregate Scores Do Not Identify Execution Profiles

The auxiliary mean averages over both retained groups. If both are nonempty, then

$$\kappa := \bar{r}_R = p\mu_{11} + (1-p)\mu_{01}, \qquad p = \frac{n_{11}}{n_{11} + n_{01}}. \tag{22}$$

The weight p is the fraction of eligible retained requirements that were observed during acquisition. The next theorem shows that the six metrics together can leave the two group means undetermined even when O, R, and B are known.

Theorem H.1 (Non-identifiability of group execution rates) *There exist two verification tables with the same reference, applicability sets, and sets O, R, and B, and equal values of KP, KR, KC, K2B, AMR, and FSR, but different values of both μ_{11} and μ_{01}. Thus the six metrics and the sets (O, R, B) do not, in general, determine the execution success rates of the retained groups.*

Proof. Let $a, b \in \mathcal{Z}$ be distinct requirements, and let every requirement apply to every instance in $\mathcal{X}$. Fix $O = \{a\}$ and $R = \{a, b\}$, with the artifact containing exactly the reference-supported claims in R. Choose success sets $S, T \subseteq \mathcal{X}$ with $0 < |S| < |T|$. In one verification table, let a succeed on S and b on T; in the other, exchange these success sets. All remaining requirements succeed on every instance in both tables.

The reference, applicability, experience, and artifact are unchanged, so KP, KR, and KC coincide. Both success sets are nonempty, giving $B = \mathcal{Z}$ in both tables. Exchanging the success sets preserves the sum of the retained requirements' success rates and the number of successful requirements on each instance, so K2B and AMR also coincide. In both tables, an instance satisfies all requirements exactly when it belongs to $S \cap T$, which preserves FSR. However, the retained group means change from $(|S|/N, |T|/N)$ to $(|T|/N, |S|/N)$ and therefore differ in both coordinates. The three-way intersection remains $O \cap R \cap B = \{a\}$. Thus neither the aggregate scores nor the fixed stage memberships determine how execution success is distributed between the retained groups. ∎

Theorem H.1 explains why the aligned profiles in Figure 17 complement the aggregate metrics. Matching requirement identities across stages makes it possible to distinguish execution of observed and retained requirements from execution of other retained requirements. The group colors reveal differences that identical aggregate scores can conceal.

Bounds on Group Execution Success. For $0 < p < 1$, Equation 22 and $0 \le \mu_{01} \le 1$ imply

$$\max\left\{0, \frac{\kappa - (1-p)}{p}\right\} \le \mu_{11} \le \min\left\{1, \frac{\kappa}{p}\right\}. \tag{23}$$

Rearranging the mixture equation and intersecting the resulting interval with $[0, 1]$ gives these bounds. Conversely, each value in the interval admits a $\mu_{01} \in [0, 1]$ satisfying the mixture equation. The interval therefore characterizes the real-valued group means compatible with fixed (p, κ).

These bounds constrain the ambiguity left by the aggregate score: when observed requirements account for most eligible retained requirements, a high auxiliary retained-requirement mean also implies a high mean success rate for that group. This implication does not identify trace-equal K2B without the applicability weights. At $p = 1$, $\mu_{11} = \kappa$; at $p = 0$, the group is empty and μ_{11} is undefined. Exchanging the groups gives the corresponding bounds for μ_{01}.

Binary Stage Correspondence. The set B also lets us examine correspondence at the level of binary membership. To relate stage sizes and pairwise overlaps to the three-way intersection, define

$$o = \frac{|O|}{M}, \qquad k = \frac{|R|}{M}, \qquad b = \frac{|B|}{M},$$
$$\alpha = \frac{|O \cap R|}{M}, \quad \beta = \frac{|R \cap B|}{M}, \quad \gamma = \frac{|O \cap B|}{M}, \quad q = \frac{|O \cap R \cap B|}{M}.$$

Here $k = \mathrm{KC}$ and $\alpha = o\,\mathrm{KR}$ when KR is defined, but β is not generally $k\bar{r}_R$ or $k\,\mathrm{K2B}$.

Lemma H.1 (Completion of binary stage summaries) *Consider a universe of M identities with fixed feasible marginals $(o, k, b, \alpha, \beta, \gamma)$. The attainable values of q are exactly the multiples of $1/M$ in $[\ell, u]$, where*

$$\begin{aligned} \ell &= \max\{0, \alpha + \gamma - o, \alpha + \beta - k, \beta + \gamma - b\}, \\ u &= \min\{\alpha, \beta, \gamma, 1 - o - k - b + \alpha + \beta + \gamma\}. \end{aligned} \tag{24}$$

All eight membership-cell counts are uniquely determined if and only if $\ell = u$.

Proof. Let p_{ijv} denote the proportion of requirements with membership indicators (i, j, v) in (O, R, B). Setting $p_{111} = q$ determines the remaining cells:

$$\begin{aligned} p_{111} &= q, & p_{110} &= \alpha - q, \\ p_{101} &= \gamma - q, & p_{011} &= \beta - q, \\ p_{100} &= o - \alpha - \gamma + q, & p_{010} &= k - \alpha - \beta + q, \\ p_{001} &= b - \beta - \gamma + q, & p_{000} &= 1 - o - k - b + \alpha + \beta + \gamma - q. \end{aligned} \tag{25}$$

The eight nonnegativity constraints give exactly the stated bounds. For any q on the $1/M$ lattice within this interval, the fixed marginal counts ensure that all Mp_{ijv} are nonnegative integers summing to M. Partitioning the identities into blocks of these sizes constructs sets with the required marginals and overlap q. Since both endpoints lie on the same lattice, the cell counts are unique precisely when the endpoints coincide. ∎

Lemma H.1 quantifies the uncertainty in three-way membership counts that remains after observing pairwise summaries. EvoTrace resolves these counts directly by preserving requirement identities across stages.

H.3 From Requirement Accuracy to Workflow Success

Per-requirement success rates do not specify how failures co-occur within workflows. At the same AMR, concentrating errors in a few tests leaves more workflows successful than spreading those errors across many tests. Branch applicability constrains these arrangements: failures of requirements on disjoint branches must occur on different instances. The sets X_a capture this structure. Define

$$f_a = |X_a| - |\{x \in X_a : a \in C_x\}|, \qquad D = \sum_{a \in \mathcal{Z}} |X_a|, \qquad F = \sum_{a \in \mathcal{Z}} f_a. \tag{26}$$

Here f_a is the failure count for requirement a, D is the total number of applicable requirement–instance pairs, and F is the total number of failures. Thus $0 \le f_a \le |X_a|$, $D > 0$, and the pair-weighted sensitivity statistic is $\mathrm{AMR}_{\mathrm{pair}} = 1 - F/D$. Reported AMR instead averages $|C_x|/|\mathcal{Z}_x|$

equally over traces. The two coincide when applicability counts are constant. Each pair in the sensitivity statistic receives equal weight; requirements with $X_a = \emptyset$ have $f_a = 0$ and contribute nothing to either total.

Theorem H.2 (Workflow success under fixed applicability) *Fix the applicability sets* (X_a) *and feasible integer failure counts* (f_a)*. Define*

$$h_{\min} = \min_{S \subseteq \mathcal{X}} \{|S| : |S \cap X_a| \geq f_a \text{ for all } a \in \mathcal{Z}\},$$
$$h_{\max} = \min_{J \subseteq \mathcal{Z}} \left\{ \sum_{a \notin J} f_a + \left| \bigcup_{a \in J} X_a \right| \right\}. \tag{27}$$

Every verification table with these applicability sets and counts satisfies

$$1 - \frac{h_{\max}}{N} \;\leq\; \text{FSR} \;\leq\; 1 - \frac{h_{\min}}{N}. \tag{28}$$

Both bounds are attained within the class of binary verification tables with applicability sets (X_a) *and failure counts* (f_a)*.*

Proof. Let $y_{a,x} \in \{0, 1\}$ indicate whether requirement a fails on instance x, with $y_{a,x} = 0$ outside X_a and $\sum_{x \in X_a} y_{a,x} = f_a$. The failed instances form the set $S(y) = \{x : \sum_a y_{a,x} > 0\}$, whose size is $h(y) = |S(y)|$. Hence $\text{FSR} = 1 - h(y)/N$.

To minimize the number of failed instances, observe that every table satisfies $|S(y) \cap X_a| \geq f_a$ for every atom. Thus $h(y) \geq h_{\min}$. Conversely, let S attain $h_{\min}$. For each atom, choose f_a distinct instances in $S \cap X_a$ and mark the corresponding pairs as failures. The set U of instances containing these failures lies in S and satisfies every covering constraint. Minimality of S rules out $|U| < |S|$, so the constructed table has exactly $h_{\min}$ failed instances.

To maximize the number of failed instances, construct a bipartite graph with atom vertices, instance vertices, and edges (a, x) whenever $x \in X_a$. Assign capacity f_a to atom a and capacity one to each instance. Selecting one failed requirement from each failed instance in a verification table gives a capacitated matching of size $h(y)$: each instance is selected once, and each atom is selected at most f_a times.

Conversely, take a maximum matching, let S be its matched instances, and mark its edges as failures. An atom with matching degree $d_a < f_a$ cannot have a neighbor outside S, since that edge would enlarge the matching. Because $|X_a| \geq f_a$, at least $f_a - d_a$ unused pairs for this atom remain within S. Marking these pairs as additional failures satisfies its prescribed count. Repeating this for every atom completes the table without changing S. Therefore, the maximum matching size equals the largest attainable value of $h(y)$.

To obtain the expression for $h_{\max}$, construct a flow network with source-to-atom capacity f_a, atom-to-applicable-instance capacity $C = F + N + 1$, and instance-to-sink capacity one. The integral max-flow/min-cut theorem equates the maximum flow with the minimum cut capacity.[1] Starting from zero, augmentation along residual source–sink paths preserves integrality and terminates after at most F augmentations. The vertices reachable from the source in the final residual graph define a cut whose capacity equals the flow value. Each integral flow selects at most one atom per instance and at most f_a instances per atom, and every such matching defines an integral flow. Thus the maximum flow equals the maximum matching size.

A minimum cut has capacity at most $F < C$ and therefore crosses no atom-to-instance edge. If J is the set of atoms on the source side, every instance in $\bigcup_{a \in J} X_a$ must lie on that side as well. For fixed J, including exactly these instances gives the smallest cut capacity, $\sum_{a \notin J} f_a + |\bigcup_{a \in J} X_a|$. Minimizing over J yields $h_{\max}$. Substituting the two attained extrema into $\text{FSR} = 1 - h(y)/N$ proves Equation 28 and the attainability of both bounds. ∎

The Role of Branch Applicability. The applicability sets constrain how failures can overlap across requirements. For any collection $J \subseteq \mathcal{Z}$ whose applicability sets are pairwise disjoint, the

[1] L. R. Ford, Jr. and D. R. Fulkerson. *Maximal Flow Through a Network*. Canadian Journal of Mathematics, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.

corresponding failures must occur on distinct instances. Consequently,

$$h_{\min} \geq \sum_{a \in J} f_a, \qquad \mathrm{FSR} \leq 1 - \frac{\sum_{a \in J} f_a}{N}.$$

This constraint can strengthen the generic bound $h(y) \geq \max_a f_a$, which does not account for branch separation. Failures of requirements shared across branches may overlap with these failed instances or affect additional ones, subject to their applicability sets. AE-SOP branch annotations therefore sharpen the relation between requirement-level accuracy and complete workflow success, as captured by the extrema in Theorem H.2.

Divergence between AMR and FSR. When every atom applies to every instance, the extrema simplify to $h_{\min} = \max_a f_a$ and $h_{\max} = \min\{N, F\}$. Even in this setting, improved requirement accuracy need not imply improved workflow success. Across checkpoints with fixed applicability, AMR increases when the total failure count F decreases, whereas FSR increases only when the number of affected instances decreases. If the remaining failures become more dispersed, they can affect more instances despite a reduction in each requirement's failure count. Reporting both metrics therefore distinguishes improved requirement satisfaction from improved workflow completion.

Interpreting Failure Concentration. Let $v_x = |\mathcal{Z}_x \setminus C_x|$ and $m_x = |\mathcal{Z}_x|$. Write $h = |\{x : v_x > 0\}|$ and, when $h > 0$, let $\bar{\ell}_{\mathrm{fail}} = h^{-1} \sum_{x: v_x > 0} v_x / m_x$ be the average violated fraction among failed traces. Then

$$1 - \mathrm{FSR} = \frac{1 - \mathrm{AMR}}{\bar{\ell}_{\mathrm{fail}}}. \tag{29}$$

This follows by separating successful traces, whose violated fraction is zero, in the trace-equal AMR average. It remains exact when branch lengths differ; substituting pooled occurrence counts would change the estimator. At fixed AMR, larger violated fractions among failures imply fewer failed workflows.

H.4 Separating Execution Changes from Retention Changes

Across checkpoints, K2B changes with both execution and retention-dependent weighting. Fix the applicability pairs $\mathcal{I} = \{(x, a) : a \in \mathcal{Z}_x\}$. For checkpoint s, let $c_s(x, a) = \mathbf{1}\{a \in C_{s,x}\}$, $m_s(x) = |R_s \cap \mathcal{Z}_x|$, and $N_s = |\{x : m_s(x) > 0\}| > 0$. Define

$$w_s(x, a) = \begin{cases} \dfrac{\mathbf{1}\{a \in R_s\}}{N_s m_s(x)}, & m_s(x) > 0, \\ 0, & m_s(x) = 0, \end{cases} \qquad \mathrm{K2B}_s = \sum_{(x,a) \in \mathcal{I}} w_s(x, a) c_s(x, a). \tag{30}$$

Proposition H.1 (An exact decomposition of K2B change) *Let $\Delta c = c_{t+1} - c_t$, $\Delta w = w_{t+1} - w_t$, $\bar{c} = (c_{t+1} + c_t)/2$, and $\bar{w} = (w_{t+1} + w_t)/2$, all defined on $\mathcal{I}$. Then*

$$\mathrm{K2B}_{t+1} - \mathrm{K2B}_t = \underbrace{\sum_{(x,a) \in \mathcal{I}} \bar{w}(x, a) \Delta c(x, a)}_{\textit{execution contribution}} + \underbrace{\sum_{(x,a) \in \mathcal{I}} \Delta w(x, a) \bar{c}(x, a)}_{\textit{retention-composition contribution}}. \tag{31}$$

Proof. For each applicability pair, expanding $\bar{w}\Delta c + \Delta w \bar{c}$ gives $w_{t+1} c_{t+1} - w_t c_t$. Summation over $\mathcal{I}$ proves the identity. ∎

The second term vanishes for a fixed retained set and fixed applicability. If execution indicators remain unchanged, the first vanishes. Retention changes also change per-trace denominators and the eligible-trace set; therefore removing requirements with low unweighted success rates does not universally increase trace-equal K2B. The sign must be computed with the actual weights. An increase in K2B alone need not imply improved execution.

Proposition H.1 links the metric to the learning curves in Appendix G.3. Aligned records distinguish execution changes from retention composition, while KC reports the accompanying reference coverage.

I WORKED EXAMPLE

Figure 8 provides a visual overview. The subsections below walk through the same order-refund scenario end to end: extracting a reference, generating instance variations, computing evidence, and deriving metrics.

I.1 AE-SOP REFERENCE EXTRACTION

The source states: "Look up the order's payment state. If captured, refund directly. Otherwise, obtain user confirmation: refund if the user agrees, or cancel the authorization if the user declines." For the worked calculation, Table 9 selects seven requirements from this fragment. The `Captured?` decision leads directly to `Refund` when true. Otherwise, `User Confirmation` obtains the user's reply, and `Refund / Cancel` performs the corresponding action. Table 9 represents this structure with five nodes and two separately scored dependency edges. This selected reference illustrates annotation and scoring rather than exhaustively specifying a production transaction workflow.

Table 9: Seven reference atoms for the order-refund example.

Atom	Graph element	Activation	Observable evidence
a_1	Look up payment state (operation)	Refund request	Tool result records the order's payment state.
a_2	Captured? (decision)	Refund request	Decision agrees with the returned payment state.
a_3	User Confirmation (operation)	Uncaptured	The agent requests confirmation and receives the user's reply.
a_4	Refund (operation)	Captured	Tool outcome records the matching refund.
a_5	Refund / Cancel (operation)	Uncaptured	Approval leads to refund; refusal leads to cancellation of the authorization. Without a reply, this check fails.
a_6	(a_2, a_4) (edge)	Captured	The captured-state decision precedes direct refund.
a_7	(a_3, a_5) (edge)	Uncaptured	The user reply precedes the corresponding update.

I.2 INSTANCE VARIATIONS

The compact JSON configurations below instantiate the reference in Table 9. `E101` and `T201` vary the entity and request wording while preserving the captured-payment branch; `T202` instead requires user confirmation for an uncaptured payment.

E101: Experience

```
{
  "configuration_id": "E101",
  "split": "experience",
  "environment_initialization":
  {
    "order_id": "E101",
    "payment_state":
      "captured"
  },
  "initial_request":
    "Please refund order E101.",
  "responses": [],
  "expected_applicable_atoms":
  [
    "a1", "a2", "a4", "a6"
  ]
}
```

T201: Held-out, same branch

```
{
  "configuration_id": "T201",
  "split": "held_out",
  "environment_initialization":
  {
    "order_id": "T201",
    "payment_state":
      "captured"
  },
  "initial_request":
    "Please return the payment
  for order T201.",
  "responses": [],
  "expected_applicable_atoms":
  [
    "a1", "a2", "a4", "a6"
  ]
}
```

T202: Held-out, different branch

```
{
  "configuration_id": "T202",
  "split": "held_out",
  "environment_initialization":
  {
    "order_id": "T202",
    "payment_state":
      "uncaptured"
  },
  "initial_request":
    "Please refund order T202.",
  "responses": [{
    "interaction_id": "a3",
    "required_content":
      "Yes, please refund it."
  }],
  "expected_applicable_atoms":
  [
    "a1", "a2", "a3", "a5", "a7"
  ]
}
```

Only `initial_request` is delivered at task entry. Payment state is obtained from the initialized environment; `responses` specifies later user content, released at the corresponding interaction point under the disclosure policy. `expected_applicable_atoms` is evaluator metadata, not agent input. A refusal such as "No, cancel the request" in `T202` would require cancellation of the authorization instead of refund.

I.3 Evidence Computation

For a captured payment, a lookup, the correct captured-state decision, and direct refund satisfy a_1, a_2, a_4, a_6; a_3, a_5, a_7 are inapplicable. Refunding before checking the state violates a_6, even if both outcomes eventually appear. Element presence alone is insufficient for dependency satisfaction.

For an uncaptured payment, lookup and the correct state decision satisfy a_1, a_2. Refunding without a user reply fails `User Confirmation` (a_3), `Refund / Cancel` (a_5), and their prerequisite edge (a_7). The recorded refund does not establish that the action agrees with the user's reply. An artifact saying "always refund directly" also omits the condition and cannot be silently restricted to captured payments.

I.4 Evidence-to-Metric Example

The seven atoms in Table 9 define the scoring universe. Test x_1 has a captured payment and activates four requirements; test x_2 has an uncaptured payment and activates five.

The experience contains a successful direct refund. Tool feedback explicitly exposes the captured-state decision and its precedence over refund. A visible policy notice states that uncaptured requests require user confirmation, exposing a_3 without executing it; the notice does not specify the reply-conditioned update a_5 or ordering edge a_7. The artifact retains lookup, the user-confirmation requirement, and direct refund for captured payments, plus an unsupported claim u. In testing, the agent completes x_1 but refunds in x_2 without obtaining a reply.

Table 10: Normalized evidence. Exp. and Art. indicate observed or explicitly retained requirements. Test entries are 1 for satisfaction, 0 for an unsatisfied applicable requirement, and – for inapplicability. The two branch prerequisites are distinct graph elements.

Atom	Requirement or claim	Exp.	Art.	x_1	x_2
a_1	Look up order payment state	1	1	1	1
a_2	Captured? Check returned payment state	1	0	1	1
a_3	User Confirmation: request and receive reply	1	1	–	0
a_4	Refund directly when captured	1	1	1	–
a_5	Refund / Cancel according to user reply	0	0	–	0
a_6	Captured-state decision precedes refund	1	0	1	–
a_7	User reply precedes the update	0	0	–	0
u	An unsupported, unmatched claim	0	1	–	–

The captured-payment test satisfies all four applicable requirements; the uncaptured-payment test satisfies two, giving $\mathrm{FSR}_t = 1/2$ and

$$\mathrm{AMR}_t = \frac{1}{2}\left(\frac{4}{4} + \frac{2}{5}\right) = \frac{7}{10}.$$

The artifact contains three reference-matched claims and the unmatched claim u; three of the five observed reference atoms are retained, giving $\mathrm{KP}_t = 3/4$, $\mathrm{KR}_t = 3/5$, and $\mathrm{KC}_t = 3/7$. Test x_1 realizes both retained applicable requirements (a_1, a_4), whereas x_2 realizes one of its two retained applicable requirements (a_1, a_3). Both traces are eligible, so

$$\mathrm{K2B}_t = \frac{1}{2}\left(\frac{2}{2} + \frac{1}{2}\right) = \frac{3}{4}.$$

Each eligible trace contributes one equally weighted retained-requirement fraction. The absent user reply leaves a_3, a_5, a_7 unsatisfied on x_2, despite the recorded refund. Test x_2 contributes 0 to FSR and $2/5$ to AMR (40% partial completion).

If the artifact additionally retained the unobserved reply-conditioned update a_5, KP would increase to $4/5$ and KC to $4/7$, while KR would remain $3/5$: a_5 matches the reference but is absent from both the KR numerator and denominator.

J Experimental Setup and Reproducibility

J.1 Evaluation Set and Configurations

Figure 9 compares the source composition of the selected 60 AE-SOPs with the full 115-AE-SOP inventory and shows their structural coverage. The selection allocates source quotas by largest remainders and matches domain, complexity-class, and structural-quintile distributions without access to performance results. The selected set retains all source and complexity categories and spans 14–587 requirements per AE-SOP; SWE contributes 41.7% of both the selected set and the full inventory after rounding.

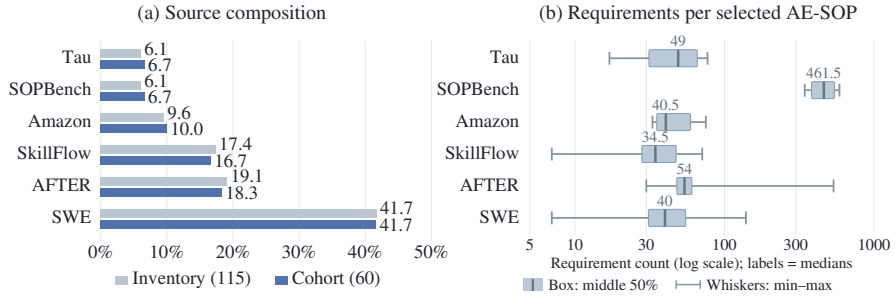


Figure 9: Selected-set source shares and requirement counts. Right: boxes span the middle 50%, gray lines and labels mark medians, and whiskers span min–max. Quartiles are linearly interpolated on counts; the axis is logarithmic.

Table 11 lists the selected source counts and experience budgets. The resulting set contains 4,008 experience and 1,924 held-out instances, totaling 5,932.

Table 11: Profile-based evaluation selection. Shares use %; counts retain exact instance budgets.

Source	AE-SOPs	Inventory share	Selection share	Experience	Test
Tau	4	6.1	6.7	272	130
SOPBench	4	6.1	6.7	1,476	738
Amazon	6	9.6	10.0	324	156
SkillFlow	10	17.4	16.7	212	96
AFTER	11	19.1	18.3	832	416
SWE	25	41.7	41.7	892	388
Total	60	100.0	100.0	4,008	1,924

Total instance budget: 5,932; requirements per AE-SOP: 14–587.
Per AE-SOP: 12–540 experience instances; 5–270 test instances.

Table 12 summarizes the evaluation protocol and model configuration. All methods share the same instance order, family budget, tools, and held-out instances within each backbone and run. Testing uses independent frozen copies of the evolution state; test feedback cannot update retained artifacts. Experience and test support are therefore aligned across methods, while experience and evolution costs remain attributable to each method and are reported separately in Appendix O. Family budgets, checkpoint-to-instance mappings, and the three order seeds are included in the released records.

Aggregation and Reproducibility. FSR averages complete execution success, whereas AMR averages each trace's correct/applicable fraction. KP, KR, and KC use their claim, observed-reference, and full-reference denominators; K2B averages retained-correct/retained-applicable fractions over eligible traces. Empty denominators remain undefined. Bootstrap draws resample families and or-

Table 12: Evaluation protocol and model configuration.

Component	Setting
Task-solving backbones	GPT-5.6-Terra; Qwen3.8-Flash-Next
Text encoder	Qwen-3-Embedding-8B; cosine similarity
Adaptive User Simulator	Claude-Sonnet-5
Trace-retrieval baseline	Top-3 prior traces by initial task description
Experience orders	Three reset runs; seeds 11, 23, 37
Checkpoints	$q \in \{0, .25, .5, .75, 1\}$; instance index $\lfloor qB_f \rfloor$
Uncertainty	2,000 paired family/order bootstrap resamples

der seeds independently with replacement, preserving the same draws for initial/final pairs; gains are computed before rounding.

Checkpoint Isolation. At each checkpoint, artifacts and persistent state are restored to the recorded evolution state. Frozen test copies cannot modify the evolution state or other test copies; the initial checkpoint supplies the reset-state comparison.

J.2 Methods, Baselines, and Adaptations

Evolution uses task-visible tool outputs and interaction history, without external rewards, gold answers, or native task-success labels. ReasoningBank and AWM-online already judge outcomes with language models; ACE also supports unlabeled evolution. All outcome-dependent updates use self-assessment. Complete method-accessible archives count as artifacts, even when retrieval injects only a subset.

No update. This baseline retains the common task-solving model, initial instructions, and tool interface, with no evolved state carried between instances. It undergoes the same checkpoints and independent held-out evaluations as the other conditions, providing a reference for performance without persistent evolution.

Trace retrieval. This baseline stores complete agent-visible execution traces without distilling claims, revising guidance, or synthesizing skills. A fixed text encoder embeds each initial task description; cosine similarity ranks prior descriptions, and the top-3 traces are injected into the context as the execution guidance.

Memento-NP retains experience cases comprising states, actions, and feedback, retrieves similar cases to inform later decisions, and updates the case bank without task-model fine-tuning. We preserve insertion, retrieval, and reward-dependent memory behavior while replacing unavailable outcome inputs with self-assessment of task-visible evidence.

A-Mem retains each interaction's original content alongside generated contextual descriptions, keywords, tags, and links to related notes. New experience can revise related notes and their organization; semantically relevant notes are retrieved for later tasks. This combines preserved experience with semantic processing rather than replacing raw content with a summary. Our configuration retains note construction, linking, evolution, and retrieval, and freezes the complete store and index at each test checkpoint.

ReasoningBank distills strategies from self-judged successful and failed trajectories and retrieves them for later tasks (Ouyang et al., 2026). We retain its extraction and consolidation loop and evaluate the base memory configuration without MaTTS.

MemP-online stores procedural memories and retrieves relevant procedures to guide new executions. Its Build–Retrieve–Update process maintains this collection, including adjustment after unsuccessful reuse. The online evolution preserves these operations but replaces the native runner's success signal with self-assessment of task-visible execution evidence (Fang et al., 2026).

ReMe distills procedural experience from contrasting traces, adapts retrieved memories to the current context, and refines the memory collection through addition, failure reflection, and utility-based deletion (Cao et al., 2026). Its native comparative recipe uses task rewards to rank trajectories.

We replace these unavailable reward inputs with self-assessments while retaining multiple-rollout comparison and refinement. All additional rollouts and assessments count toward evolution cost.

ExpeL retains both textual insights and prior task demonstrations (Zhao et al., 2024). It extracts reusable guidance by comparing successful and unsuccessful experiences and reviewing successful trajectories, then loads the complete current insights/rules together with retrieved successful demonstrations for subsequent solving. We preserve these complementary components while replacing unavailable native success labels with self-assessment.

Dynamic Cheatsheet maintains a revisable textual document containing strategies, useful facts, and reusable reasoning patterns accumulated across tasks. It loads the complete current cheatsheet together with retrieved historical input/output examples (Suzgun et al., 2026).

ACE maintains a playbook through generator, reflector, and curator roles (Zhang et al., 2026a). We preserve the full playbook, helpful/harmful counters, and incremental updates; the reflector uses task-visible execution evidence and self-assessment, without gold labels.

AWM-online uses a language-model evaluator to select self-predicted successful traces, abstracts reusable action patterns, and adds them to the solver's memory (Wang et al., 2025). We preserve this online loop and restrict its evaluator to task-visible evidence.

EvoSkill proposes reusable skills from execution failures, evaluates candidates, and retains a frontier of solutions (Alzubi et al., 2026). Skill specifications evolve while candidates inherit the parent system prompt, tool configuration, and output format. Skill discovery and frontier maintenance are preserved.

Memento-Skills retains skills as structured skill specifications with helper scripts and prompts, selected through a behavior-oriented router (Zhou et al., 2026). Failure attribution identifies the responsible skill for targeted revision; low running utility can trigger restructuring or new-skill discovery. Self-generated test cases gate updates, with rejected changes rolled back. We deploy this read–execute–reflect–write loop and its utility, routing, judging, and validation state. Gold-dependent judging is replaced with self-assessment.

K Detailed Metric Statistics

This section supplements the main-text results with detailed per-checkpoint statistics and learning curves. The checkpoint table and FSR curves use Qwen3.8-Flash-Next to complement GPT in the main text; AMR, KP, KR, KC, K2B, and the combined uncertainty table cover both backbones.

Throughout, Avg denotes an equal-method mean. Class abbreviations are Ctrl (controls), CN (case memories), KB (knowledge banks), GW (guidance documents), and SL (skill libraries), containing 2/2/3/4/2 methods, respectively; Evolving Avg excludes the two controls, and Overall Avg includes all 13 conditions. Undefined entries are excluded per metric, not replaced by zero (overall KP and K2B use 12 methods; Ctrl uses one). Pale blue highlights group averages and selected individual cells discussed in the text; deeper blue marks Overall Avg. Highlights flag informative contrasts, not statistical significance.

K.1 Execution Results at Checkpoints

Table 13 reports Qwen FSR and AMR at all five checkpoints. Improvement is not monotone: ReMe's FSR decreases from 58.5% to 58.4% over the last interval while AMR rises from 90.6% to 90.8%. At the endpoint, ExpeL has higher FSR than ReasoningBank (60.1% versus 59.7%) but lower AMR (91.2% versus 91.3%), illustrating why complete execution success and partial requirement completion are reported separately.

Table 13: Qwen checkpoint FSR / AMR (%).

Class	Method	0%	25%	50%	75%	100%
Ctrl	No update	45.8 / 85.9	45.6 / 86.3	45.2 / 86.0	47.7 / 86.4	47.3 / 86.5
Ctrl	Trace retrieval	45.8 / 85.9	47.6 / 87.0	47.5 / 87.0	50.5 / 87.7	50.7 / 87.9
Ctrl	**Ctrl Avg (2)**	**45.8 / 85.9**	**46.6 / 86.6**	**46.4 / 86.5**	**49.1 / 87.1**	**49.0 / 87.2**
CN	Memento-NP	45.8 / 85.9	50.9 / 88.3	53.4 / 89.1	55.8 / 89.7	56.0 / 89.8
CN	A-Mem	45.8 / 85.9	48.9 / 87.5	49.4 / 87.6	52.0 / 88.1	52.4 / 88.4
CN	**CN Avg (2)**	**45.8 / 85.9**	**49.9 / 87.9**	**51.4 / 88.4**	**53.9 / 88.9**	**54.2 / 89.1**
KB	ReasoningBank	45.8 / 85.9	53.6 / 89.4	56.3 / 90.2	59.2 / 90.9	59.7 / 91.3
KB	MemP-online	45.8 / 85.9	47.1 / 86.8	47.8 / 87.0	49.8 / 87.2	49.9 / 87.4
KB	ReMe	45.8 / 85.9	51.8 / 88.7	55.2 / 89.8	58.5 / 90.6	58.4 / 90.8
KB	**KB Avg (3)**	**45.8 / 85.9**	**50.8 / 88.3**	**53.1 / 89.0**	**55.8 / 89.6**	**56.0 / 89.8**
GW	ExpeL	45.8 / 85.9	54.8 / 89.6	57.1 / 90.4	59.1 / 90.8	60.1 / 91.2
GW	D. Cheatsheet	45.8 / 85.9	46.0 / 86.3	46.3 / 86.3	48.7 / 86.8	49.3 / 87.3
GW	ACE	45.8 / 85.9	53.1 / 89.1	59.6 / 91.1	64.5 / 92.5	67.2 / 93.3
GW	AWM-online	45.8 / 85.9	51.4 / 88.5	54.3 / 89.5	57.2 / 90.3	58.3 / 90.7
GW	**GW Avg (4)**	**45.8 / 85.9**	**51.3 / 88.4**	**54.3 / 89.3**	**57.4 / 90.1**	**58.7 / 90.6**
SL	EvoSkill	45.8 / 85.9	50.5 / 88.1	54.9 / 89.8	58.8 / 90.8	61.0 / 91.5
SL	Memento-Skills	45.8 / 85.9	50.5 / 88.2	55.7 / 89.9	60.1 / 91.1	62.1 / 92.0
SL	**SL Avg (2)**	**45.8 / 85.9**	**50.5 / 88.2**	**55.3 / 89.8**	**59.4 / 91.0**	**61.5 / 91.7**
All	**Evolving Avg (11)**	**45.8 / 85.9**	**50.8 / 88.2**	**53.6 / 89.1**	**56.7 / 89.9**	**57.7 / 90.3**
All	**Overall Avg (13)**	**45.8 / 85.9**	**50.1 / 88.0**	**52.5 / 88.7**	**55.5 / 89.5**	**56.3 / 89.8**

Table 14 combines both backbones' endpoint metrics and from-initial gains with 95% paired-bootstrap confidence intervals. Several GPT gain intervals include zero (e.g., trace retrieval and Dynamic Cheatsheet), so an apparent positive endpoint difference does not establish a reliable gain. Qwen shows more clearly positive from-initial changes, but these intervals do not test pairwise superiority between methods. The eleven evolving methods average FSR gains of 8.8 pp on GPT and 11.9 pp on Qwen; including controls yields Overall Avg gains of 7.5 and 10.6 pp.

K.2 Learning Curves across Metrics

Figure 10 reports Qwen FSR at the five checkpoints, grouped by artifact class. Execution success is not monotone: trace retrieval dips slightly between 25% and 50%, and ReMe dips in the last interval. ACE improves throughout, whereas the remaining guidance methods follow divergent trajectories despite sharing the same artifact category. This within-class variation reinforces the main-text finding that artifact form alone does not determine execution gains; the specific mechanism by which retained knowledge is selected and applied matters at least as much as the representation it takes.

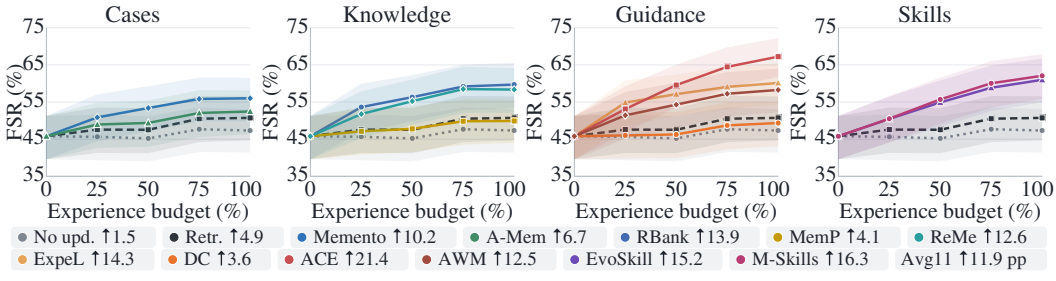


Figure 10: Qwen FSR learning curves with pointwise 95% bootstrap intervals.

Figure 11 reports AMR for both backbones on the same scale. AMR remains higher than FSR on both backbones, reflecting partial requirement completion in unsuccessful executions. Final Overall Avg AMR is 89.3% on GPT and 89.8% on Qwen. Qwen ReMe's late AMR rise despite a slight FSR fall illustrates that these two execution metrics capture different facets of procedural execution: an agent can satisfy more individual requirements on average even as the fraction of fully correct executions stagnates or declines, because additional requirements are realized in traces that still contain at least one remaining violation.

Table 14: Endpoint metrics and gains with 95% paired-bootstrap intervals.

Method	Final FSR (%)	ΔFSR (pp)	Final AMR (%)	ΔAMR (pp)
GPT-5.6-Terra				
No update	45.5 [39.2, 51.8]	-1.3 [-4.9, 2.6]	86.0 [84.3, 87.6]	-0.1 [-1.5, 1.3]
Trace retrieval	48.2 [41.7, 54.6]	+1.4 [-2.6, 6.0]	87.2 [85.5, 88.9]	+1.1 [-0.5, 2.8]
Ctrl Avg (2)	**46.8 [40.4, 53.2]**	**+0.1 [-3.7, 4.3]**	**86.6 [84.9, 88.2]**	**+0.5 [-1.0, 2.0]**
Memento-NP	53.4 [46.9, 59.5]	+6.6 [3.1, 10.5]	88.9 [87.5, 90.2]	+2.8 [1.4, 4.3]
A-Mem	49.8 [43.3, 56.2]	+3.0 [-1.0, 7.1]	87.6 [86.0, 89.1]	+1.5 [0.0, 3.0]
CN Avg (2)	**51.6 [45.2, 57.8]**	**+4.8 [1.2, 8.8]**	**88.3 [86.8, 89.6]**	**+2.1 [0.8, 3.6]**
ReasoningBank	57.9 [51.0, 64.2]	+11.1 [7.4, 14.9]	90.8 [89.6, 91.9]	+4.6 [3.4, 5.9]
MemP-online	47.9 [41.4, 54.2]	+1.1 [-2.9, 5.5]	86.9 [85.1, 88.4]	+0.7 [-0.8, 2.3]
ReMe	57.3 [50.6, 63.8]	+10.5 [6.4, 14.7]	90.6 [89.2, 91.7]	+4.4 [2.9, 5.9]
KB Avg (3)	**54.3 [47.8, 60.7]**	**+7.5 [3.8, 11.5]**	**89.4 [88.0, 90.6]**	**+3.3 [1.9, 4.7]**
ExpeL	57.8 [51.1, 64.0]	+11.0 [7.3, 15.2]	90.8 [89.5, 91.9]	+4.6 [3.2, 6.0]
D. Cheatsheet	47.5 [41.0, 53.7]	+0.7 [-2.8, 4.6]	86.6 [85.0, 88.0]	+0.4 [-0.8, 1.7]
ACE	65.0 [59.2, 70.4]	+18.2 [14.3, 22.4]	92.8 [91.8, 93.7]	+6.6 [5.2, 8.0]
AWM-online	55.5 [49.2, 61.5]	+8.7 [4.8, 12.8]	90.0 [88.7, 91.1]	+3.8 [2.4, 5.2]
GW Avg (4)	**56.5 [50.2, 62.3]**	**+9.7 [6.2, 13.6]**	**90.0 [88.8, 91.1]**	**+3.9 [2.6, 5.1]**
EvoSkill	59.6 [53.0, 65.9]	+12.9 [8.9, 17.5]	91.3 [90.1, 92.5]	+5.2 [3.8, 6.6]
Memento-Skills	59.9 [53.6, 66.0]	+13.1 [9.5, 17.1]	91.4 [90.2, 92.5]	+5.2 [3.8, 6.7]
SL Avg (2)	**59.8 [53.2, 65.8]**	**+13.0 [9.2, 17.3]**	**91.4 [90.1, 92.4]**	**+5.2 [3.8, 6.6]**
Evolving Avg (11)	**55.6 [49.3, 61.7]**	**+8.8 [5.3, 12.7]**	**89.8 [88.5, 90.9]**	**+3.6 [2.3, 5.0]**
Overall Avg (13)	**54.3 [47.9, 60.3]**	**+7.5 [3.9, 11.3]**	**89.3 [87.9, 90.5]**	**+3.1 [1.8, 4.5]**
Qwen3.8-Flash-Next				
No update	47.3 [41.4, 53.1]	+1.5 [-1.0, 4.2]	86.5 [85.1, 87.8]	+0.6 [-0.5, 1.7]
Trace retrieval	50.7 [44.7, 56.5]	+4.9 [2.3, 7.7]	87.9 [86.7, 89.1]	+1.9 [0.9, 3.1]
Ctrl Avg (2)	**49.0 [43.1, 54.8]**	**+3.2 [0.7, 5.9]**	**87.2 [85.9, 88.4]**	**+1.2 [0.2, 2.4]**
Memento-NP	56.0 [49.8, 61.5]	+10.2 [7.4, 13.2]	89.8 [88.8, 90.8]	+3.9 [2.7, 5.1]
A-Mem	52.4 [46.3, 58.2]	+6.7 [3.6, 9.7]	88.4 [87.2, 89.6]	+2.5 [1.4, 3.7]
CN Avg (2)	**54.2 [48.2, 59.7]**	**+8.4 [5.8, 11.2]**	**89.1 [88.1, 90.2]**	**+3.2 [2.1, 4.4]**
ReasoningBank	59.7 [53.4, 65.6]	+13.9 [10.8, 17.1]	91.3 [90.3, 92.3]	+5.4 [4.3, 6.6]
MemP-online	49.9 [44.0, 55.6]	+4.1 [1.1, 7.1]	87.4 [86.0, 88.7]	+1.4 [0.3, 2.6]
ReMe	58.4 [52.1, 64.1]	+12.6 [10.0, 15.3]	90.8 [89.8, 91.9]	+4.9 [3.8, 6.0]
KB Avg (3)	**56.0 [49.9, 61.7]**	**+10.2 [7.5, 12.9]**	**89.8 [88.7, 90.9]**	**+3.9 [2.9, 5.0]**
ExpeL	60.1 [54.4, 65.5]	+14.3 [11.3, 17.3]	91.2 [90.2, 92.2]	+5.3 [4.1, 6.5]
D. Cheatsheet	49.3 [43.1, 55.0]	+3.6 [0.9, 6.2]	87.3 [86.0, 88.5]	+1.3 [0.2, 2.4]
ACE	67.2 [61.8, 72.3]	+21.4 [18.5, 24.4]	93.3 [92.5, 94.2]	+7.4 [6.2, 8.7]
AWM-online	58.3 [52.2, 64.0]	+12.5 [9.7, 15.4]	90.7 [89.6, 91.7]	+4.7 [3.6, 5.9]
GW Avg (4)	**58.7 [53.0, 64.1]**	**+13.0 [10.3, 15.6]**	**90.6 [89.6, 91.6]**	**+4.7 [3.6, 5.8]**
EvoSkill	61.0 [55.0, 66.7]	+15.2 [12.1, 18.3]	91.5 [90.5, 92.6]	+5.6 [4.5, 6.8]
Memento-Skills	62.1 [55.9, 67.9]	+16.3 [13.4, 19.3]	92.0 [90.9, 93.1]	+6.0 [4.9, 7.3]
SL Avg (2)	**61.5 [55.5, 67.3]**	**+15.7 [12.8, 18.8]**	**91.7 [90.7, 92.8]**	**+5.8 [4.7, 7.0]**
Evolving Avg (11)	**57.7 [51.6, 63.2]**	**+11.9 [9.3, 14.4]**	**90.3 [89.4, 91.3]**	**+4.4 [3.4, 5.5]**
Overall Avg (13)	**56.3 [50.4, 61.9]**	**+10.6 [8.0, 13.1]**	**89.8 [88.8, 90.9]**	**+3.9 [2.9, 5.0]**

Figure 12 compares KP across both backbones. Precision does not necessarily improve with experience: case-memory Avg KP falls to 76.1% on GPT and 76.2% on Qwen, while skill-library Avg remains above 99%. Local fluctuations also persist: Memento-Skills precision drops between 50% and 75% on both backbones. The contrast between case memories and skill libraries reflects a structural difference: case memories preserve raw traces whose content may diverge from reference requirements, whereas skill libraries distill selective procedural claims that are more likely to align with the reference but cover fewer requirements overall.

Figure 13 reports KR on both backbones, using requirements encountered during experience as the denominator. Complete case-memory archives retain 100% of observed requirements, whereas final skill-library Avg KR is 31.1% on GPT and 29.7% on Qwen despite high KP. Retention is also locally nonmonotone: ReMe declines between 50% and 75% on both backbones. Together with the KP curves, these results show that precision and retention measure different properties of the same artifact: a method can maintain high precision while losing previously retained requirements

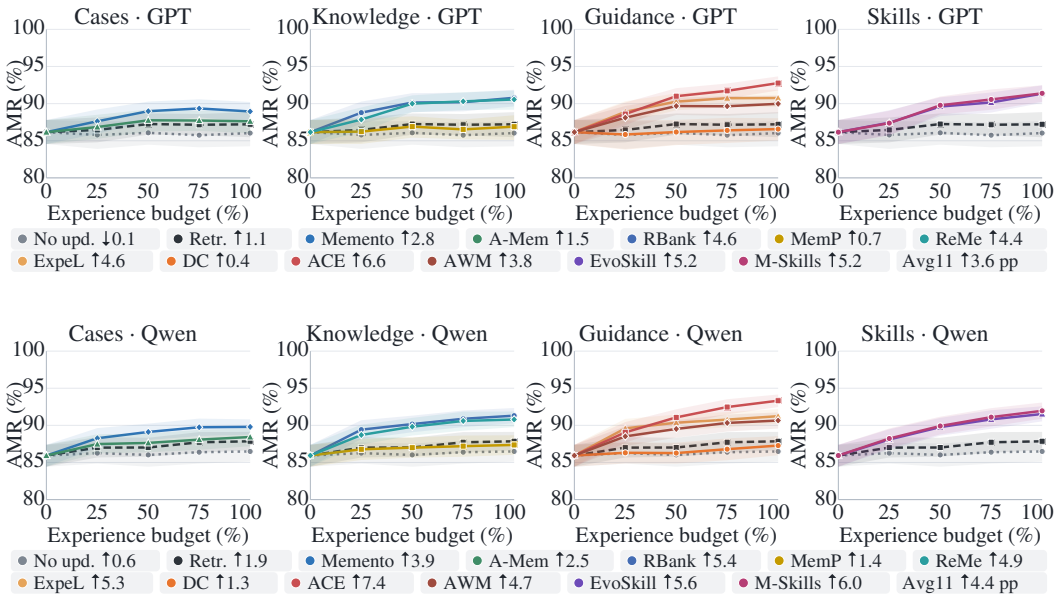


Figure 11: AMR learning curves for GPT (top) and Qwen (bottom).

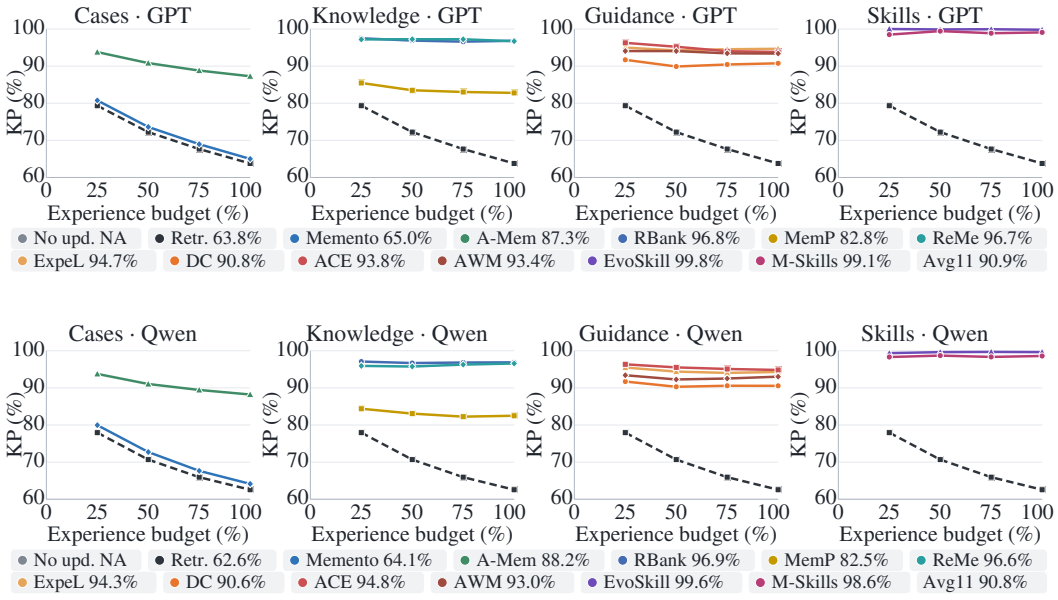


Figure 12: KP learning curves for GPT (top) and Qwen (bottom).

through consolidation or pruning, and conversely retain all observed content while diluting precision with irrelevant material.

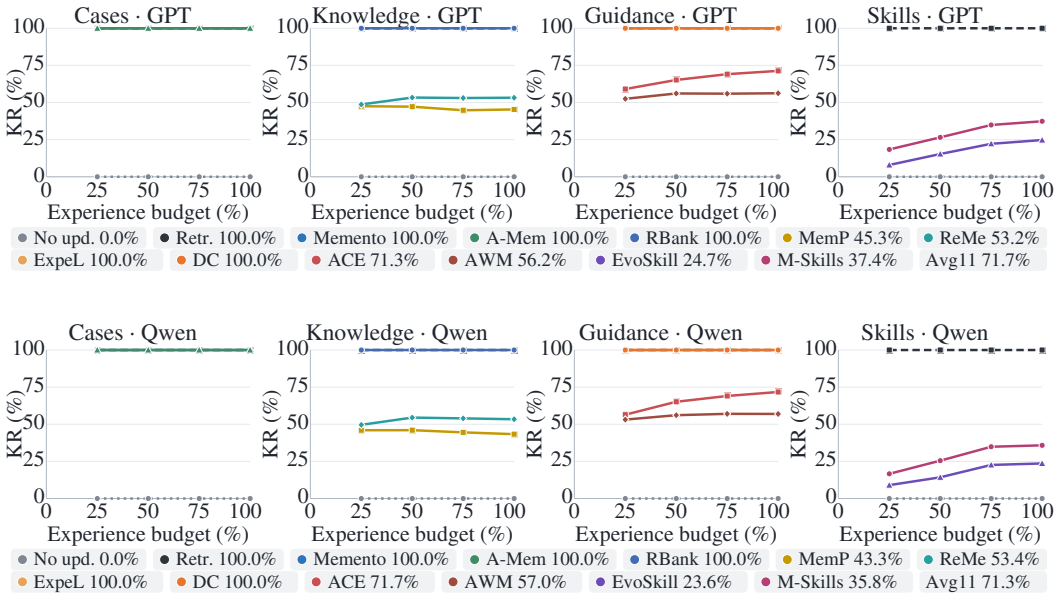


Figure 13: KR learning curves for GPT (top) and Qwen (bottom).

Figure 14 compares KC across both backbones. Unlike KR, KC uses the complete reference set, including requirements not yet encountered during experience. Broad case-memory and guidance stores attain greater explicit coverage than compact skill libraries on both backbones: final skill-library Avg KC is only 17.2% on GPT and 16.6% on Qwen despite precision above 99%; Overall Avg KC is 35.9% on both. Coverage thus does not reproduce either precision or execution-success rankings. This dissociation arises because skill libraries distill a narrow, high-fidelity subset of the procedure, whereas case memories and guidance documents preserve wider but less selective content. The implication is that evaluating evolution solely by the breadth of retained knowledge would overestimate methods that accumulate content without necessarily improving execution.

Figure 15 presents K2B as endpoint bars for both backbones. K2B is comparatively stationary for several methods: Dynamic Cheatsheet changes by only +0.2 pp on GPT and +0.4 pp on Qwen, while ACE gains +3.7 and +3.2 pp. Final Overall Avg K2B is 90.8% and 91.2%; high realization conditional on retention can coexist with low coverage of the full procedure. This pattern highlights a fundamental trade-off: methods that retain selectively tend to realize what they keep, but their narrow coverage limits the procedure fraction they can improve, whereas methods with broad retention face the harder problem of reliably applying a larger body of knowledge during execution.

Observed Evidence along the Learning Curve. Figure 16 complements the metric curves with experience-observed reference coverage on both backbones. Equal experience progress does not produce equal evidence coverage: GPT ACE observes 56.2% of references at the endpoint versus 46.9% for trace retrieval. This gap separates what experience makes observable from what the method subsequently retains. Because ACE's multi-role evolution actively explores alternative execution strategies, it encounters a broader set of procedural branches than passive trace accumulation. More available experience therefore does not reveal the full procedure, and different evolution mechanisms extract different amounts of evidence from the same experience budget.

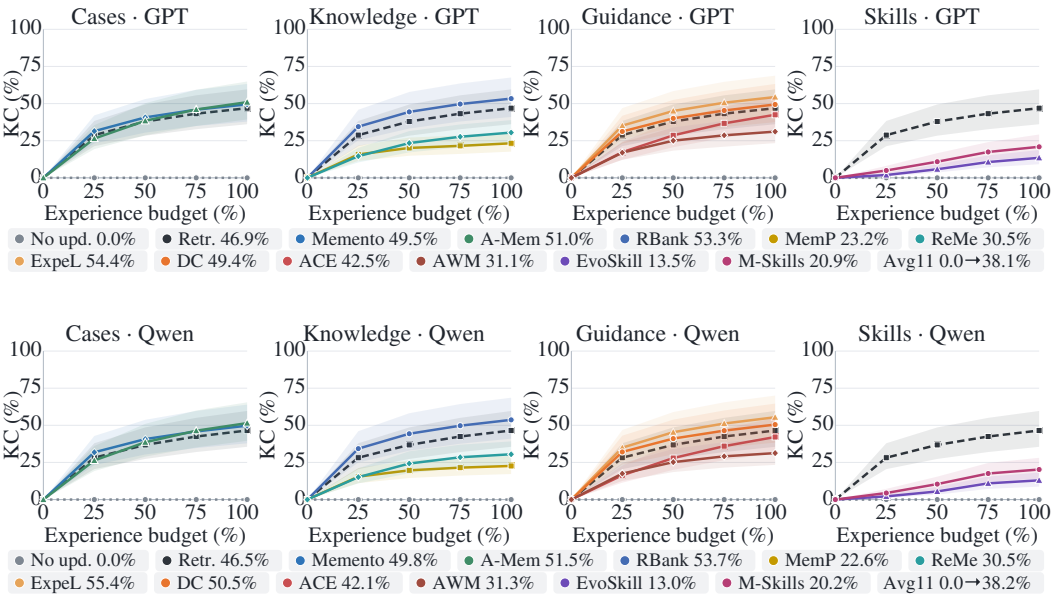


Figure 14: KC learning curves for GPT (top) and Qwen (bottom).

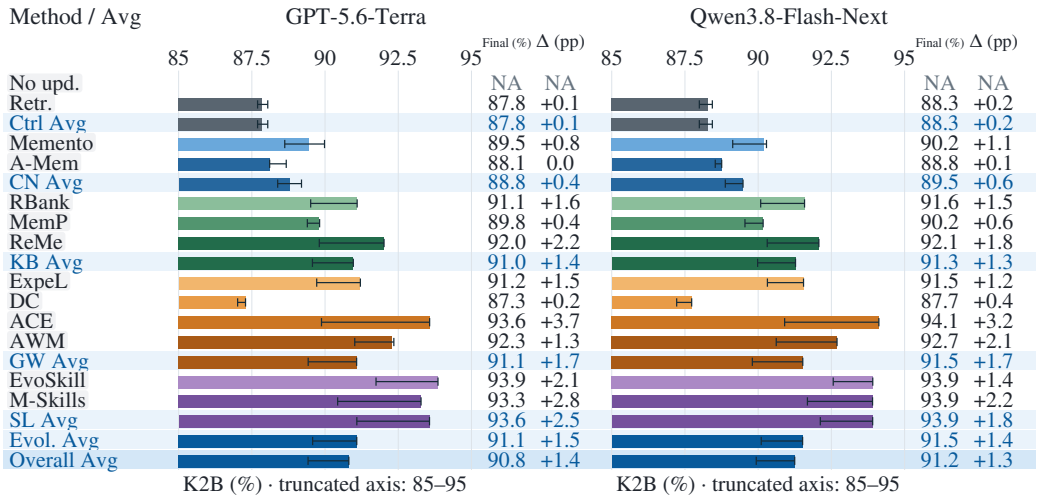


Figure 15: K2B endpoint comparison and checkpoint variation on both backbones.

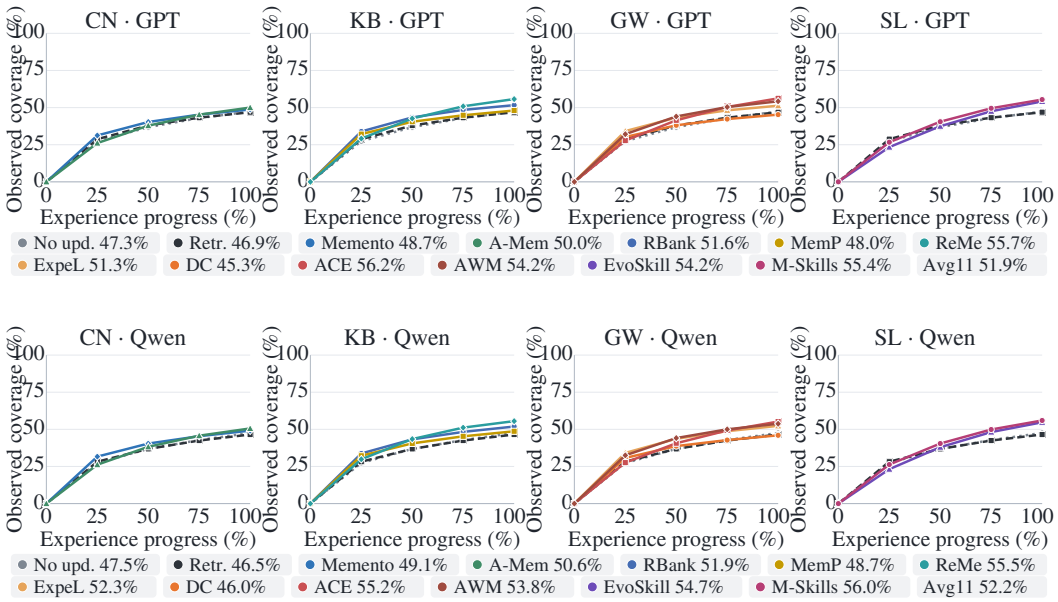


Figure 16: Observed reference coverage along experience (%).

K.3 Quantitative Learning Dynamics

Table 15 summarizes learning dynamics per family/order run, averaged equally across 180 runs. AULC integrates from-initial improvement over actual attempted-instance counts, normalized by budget. Early/late speeds divide FSR changes by attempts over the 0–50% and 50–100% experience intervals. Declines counts negative FSR intervals; Avg $D_{\max}$ averages each run's largest one-interval drop (zero if none).

ReasoningBank exceeds EvoSkill in FSR AULC on GPT (7.75 versus 6.70) and Qwen (9.89 versus 8.75), despite lower final FSR. This reversal arises because ReasoningBank front-loads its gains: GPT ReasoningBank achieves early speed of 0.61 pp/instance then slows to 0.26, while EvoSkill accelerates from 0.38 to 0.50. Qwen EvoSkill instead slows from 0.74 to 0.63, suggesting that the acceleration pattern on GPT reflects a backbone-specific interaction rather than an intrinsic property of skill-based evolution. Pooled smoothness can also mask regressions: GPT Dynamic Cheatsheet has negative AULC (-0.44 pp), 40.0% declining intervals, and a mean maximum drop of 15.51 pp. Because its guidance document accumulates conflicting strategies from heterogeneous procedure families, later updates can overwrite previously effective content, producing net regression despite continued experience.

Table 15: Learning dynamics summary on both backbones.

Method	FSR AULC	AMR AULC	Early s	Late s	Declines	Avg $D_{\max}$
GPT-5.6-Terra						
No update	-1.54	-0.18	-0.24	-0.08	44.4	15.81
Trace retrieval	0.73	0.82	-0.01	-0.07	41.5	15.32
Ctrl Avg (2)	**-0.41**	**0.32**	**-0.12**	**-0.07**	**43.0**	**15.57**
Memento-NP	4.79	2.29	0.38	0.07	39.7	14.33
A-Mem	1.98	1.19	0.16	-0.05	42.5	14.88
CN Avg (2)	**3.38**	**1.74**	**0.27**	**0.01**	**41.1**	**14.60**
ReasoningBank	7.75	3.44	0.61	0.26	36.5	14.48
MemP-online	0.37	0.43	0.02	-0.04	43.1	15.46
ReMe	6.87	3.12	0.62	0.18	38.6	13.91
KB Avg (3)	**4.99**	**2.33**	**0.41**	**0.13**	**39.4**	**14.62**
ExpeL	8.38	3.60	0.72	0.15	35.8	13.60
D. Cheatsheet	-0.44	0.04	-0.15	0.04	40.0	15.51
ACE	10.56	4.23	0.85	0.71	31.5	11.70
AWM-online	6.05	2.81	0.48	0.18	38.3	14.19
GW Avg (4)	**6.14**	**2.67**	**0.48**	**0.27**	**36.4**	**13.75**
EvoSkill	6.70	3.00	0.38	0.50	35.6	12.48
Memento-Skills	7.11	3.09	0.39	0.49	36.2	12.42
SL Avg (2)	**6.91**	**3.04**	**0.39**	**0.50**	**35.9**	**12.45**
Evolving Avg (11)	**5.46**	**2.48**	**0.41**	**0.23**	**38.0**	**13.91**
Overall Avg (13)	**4.56**	**2.14**	**0.32**	**0.18**	**38.8**	**14.16**
Qwen3.8-Flash-Next						
No update	0.32	0.24	0.01	0.24	42.2	14.88
Trace retrieval	2.61	1.23	0.21	0.33	40.0	14.62
Ctrl Avg (2)	**1.47**	**0.73**	**0.11**	**0.28**	**41.1**	**14.75**
Memento-NP	7.10	2.95	0.79	0.27	36.7	13.28
A-Mem	3.97	1.68	0.38	0.34	38.8	14.47
CN Avg (2)	**5.54**	**2.32**	**0.59**	**0.31**	**37.7**	**13.88**
ReasoningBank	9.89	4.02	1.01	0.44	34.0	12.72
MemP-online	2.39	0.99	0.29	0.20	41.7	14.54
ReMe	8.66	3.51	0.93	0.36	35.0	12.46
KB Avg (3)	**6.98**	**2.84**	**0.74**	**0.34**	**36.9**	**13.24**
ExpeL	10.38	4.07	1.16	0.33	34.2	13.21
D. Cheatsheet	1.19	0.52	0.18	0.28	39.2	15.14
ACE	12.79	4.81	1.25	0.97	26.9	10.28
AWM-online	7.99	3.32	0.86	0.41	35.4	13.61
GW Avg (4)	**8.09**	**3.18**	**0.86**	**0.50**	**33.9**	**13.06**
EvoSkill	8.75	3.56	0.74	0.63	32.9	11.43
Memento-Skills	9.45	3.79	0.82	0.66	31.5	11.82
SL Avg (2)	**9.10**	**3.68**	**0.78**	**0.65**	**32.2**	**11.62**
Evolving Avg (11)	**7.51**	**3.02**	**0.76**	**0.44**	**35.1**	**13.00**
Overall Avg (13)	**6.58**	**2.67**	**0.66**	**0.42**	**36.0**	**13.27**

K.4 Numerical Supplement to Main-Text Diagnostic Profiles

This subsection provides the numerical values behind the main-text diagnostic heatmap. The combined endpoint KP/KR/KC/K2B table is presented in §K.5; reference-state accounting and failure burden are developed separately in Appendix L.

Figure 17 reports the exact diagnostic values for all 13 conditions at the final checkpoint. On GPT, decision-related artifact misses (A) affect 63.7% of MemP-online's failed executions versus 8.5% for ReasoningBank, yet execution misses (X) remain comparable at 65.0% versus 61.3%. Qwen shows the same pattern (65.7% versus 7.9% for A; 66.7% versus 62.5% for X). These are overlapping execution proportions, not a partition of failures; A is nested in X. The gap between artifact-miss and execution-miss prevalence indicates that ReasoningBank's failures stem primarily from execution-time reuse rather than retention, whereas MemP-online's failures are heavily attributable to absent decision knowledge in its artifacts. This distinction is actionable: the two methods require different intervention targets despite similar overall execution-miss rates.

(a) GPT

Learning-process metrics (%; gains in pp)

Method	ΔFSR	ΔAMR	KP	KR	KC	K2B
	≤0 >16	≤0 >6	60 100	0 100	0 60	85 95
No upd.	-1.3	-0.1	NA	0.0*	0.0*	NA
Retr.	+1.4	+1.1	63.8	100.0	46.9	87.8
Ctrl Avg	+0.1	+0.5	63.8	50.0	23.4	87.8
Memento	+6.6	+2.8	65.0	100.0	49.5	89.5
A-Mem	+3.0	+1.5	87.3	100.0	51.0	88.1
CN Avg	+4.8	+2.1	76.1	100.0	50.2	88.8
RBank	+11.1	+4.6	96.8	100.0	53.3	91.1
MemP	+1.1	+0.7	82.8	45.3	23.2	89.8
ReMe	+10.5	+4.4	96.7	53.2	30.5	92.0
KB Avg	+7.5	+3.3	92.1	66.2	35.7	91.0
ExpeL	+11.0	+4.6	94.7	100.0	54.4	91.2
DC	+0.7	+0.4	90.8	100.0	49.4	87.3
ACE	+18.2	+6.6	93.8	71.3	42.5	93.6
AWM	+8.7	+3.8	93.4	56.2	31.1	92.3
GW Avg	+9.7	+3.9	93.2	81.9	44.3	91.1
EvoSkill	+12.9	+5.2	99.8	24.7	13.5	93.9
M-Skills	+13.1	+5.2	99.1	37.4	20.9	93.3
SL Avg	+13.0	+5.2	99.4	31.1	17.2	93.6
Evol. Avg	+8.8	+3.6	90.9	71.7	38.1	91.1
Overall Avg	+7.5	+3.1	88.7	68.3	35.9	90.8

Requirement misses among failed workflows (%)

Method	Op. A	Op. X	Dec. A	Dec. X	Dep. A	Dep. X
	3 83	73 83	7 67	60 71	22 66	59 66
No upd.	81.7	81.7	64.1	64.1	64.8	64.8
Retr.	4.7	80.7	15.9	64.2	30.9	64.2
Ctrl Avg	43.2	81.2	40.0	64.2	47.9	64.5
Memento	5.1	79.6	13.1	62.0	28.7	64.2
A-Mem	3.7	80.0	12.7	64.3	26.3	63.9
CN Avg	4.4	79.8	12.9	63.1	27.5	64.0
RBank	6.5	78.9	8.5	61.3	23.7	62.4
MemP	25.1	80.1	63.7	65.0	55.6	64.9
ReMe	30.1	79.2	34.6	62.1	47.8	62.7
KB Avg	20.6	79.4	35.6	62.8	42.3	63.3
ExpeL	5.8	79.5	10.2	60.0	23.7	63.6
DC	5.7	79.6	18.4	68.1	29.5	65.5
ACE	13.4	77.1	24.7	60.1	40.7	62.8
AWM	18.5	75.9	62.7	66.7	41.7	60.5
GW Avg	10.8	78.0	29.0	63.8	33.9	63.1
EvoSkill	51.9	76.9	54.6	64.0	59.4	63.0
M-Skills	40.9	76.9	45.9	64.0	53.7	62.3
SL Avg	46.4	76.9	50.3	64.0	56.6	62.6
Evol. Avg	18.8	78.5	31.7	63.4	39.2	63.3
Overall Avg	22.5	78.9	33.0	63.5	40.5	63.4

A: artifact miss plus execution miss. X: execution miss.

(b) Qwen

Learning-process metrics (%; gains in pp)

Method	ΔFSR	ΔAMR	KP	KR	KC	K2B
	≤0 >16	≤0 >6	60 100	0 100	0 60	85 95
No upd.	+1.5	+0.6	NA	0.0*	0.0*	NA
Retr.	+4.9	+1.9	62.6	100.0	46.5	88.3
Ctrl Avg	+3.2	+1.2	62.6	50.0	23.3	88.3
Memento	+10.2	+3.9	64.1	100.0	49.8	90.2
A-Mem	+6.7	+2.5	88.2	100.0	51.5	88.8
CN Avg	+8.4	+3.2	76.2	100.0	50.6	89.5
RBank	+13.9	+5.4	96.9	100.0	53.7	91.6
MemP	+4.1	+1.4	82.5	43.3	22.6	90.2
ReMe	+12.6	+4.9	96.6	53.4	30.5	92.1
KB Avg	+10.2	+3.9	92.0	65.6	35.6	91.3
ExpeL	+14.3	+5.3	94.3	100.0	55.4	91.5
DC	+3.6	+1.3	90.6	100.0	50.5	87.7
ACE	+21.4	+7.4	94.8	71.7	42.1	94.1
AWM	+12.5	+4.7	93.0	57.0	31.3	92.7
GW Avg	+13.0	+4.7	93.2	82.2	44.8	91.5
EvoSkill	+15.2	+5.6	99.6	23.6	13.0	93.9
M-Skills	+16.3	+6.0	98.6	35.8	20.2	93.9
SL Avg	+15.7	+5.8	99.1	29.7	16.6	93.9
Evol. Avg	+11.9	+4.4	90.8	71.3	38.2	91.5
Overall Avg	+10.6	+3.9	88.5	68.1	35.9	91.2

Requirement misses among failed workflows (%)

Method	Op. A	Op. X	Dec. A	Dec. X	Dep. A	Dep. X
	3 83	73 83	7 67	60 71	22 66	59 66
No upd.	82.0	82.0	64.5	64.5	65.0	65.0
Retr.	4.8	80.8	14.2	65.2	29.4	64.6
Ctrl Avg	43.4	81.4	39.4	64.9	47.2	64.8
Memento	4.9	78.3	11.0	65.5	26.8	62.6
A-Mem	3.3	79.6	11.5	65.4	25.3	64.7
CN Avg	4.1	79.0	11.2	65.4	26.0	63.7
RBank	5.6	78.2	7.9	62.5	22.2	60.6
MemP	24.1	79.4	65.7	66.7	56.2	65.6
ReMe	26.6	77.9	36.0	63.8	47.8	62.5
KB Avg	18.8	78.5	36.5	64.3	42.1	62.9
ExpeL	4.3	77.8	9.0	62.6	22.4	61.6
DC	5.5	77.3	14.6	69.7	28.4	64.4
ACE	14.4	74.2	26.2	63.2	40.4	62.2
AWM	15.4	73.1	66.8	70.2	39.5	59.5
GW Avg	9.9	75.6	29.2	66.4	32.7	61.9
EvoSkill	49.9	76.1	56.4	65.6	59.2	63.1
M-Skills	40.7	76.9	50.6	65.2	55.1	63.0
SL Avg	45.3	76.5	53.5	65.4	57.1	63.1
Evol. Avg	17.7	77.2	32.3	65.5	38.5	62.7
Overall Avg	21.7	77.8	33.4	65.4	39.8	63.0

A: artifact miss plus execution miss. X: execution miss.

Figure 17: Diagnostic values for all 13 conditions at the final checkpoint.

K.5 Tabulated Artifact Metrics

Table 16 aligns final KP, KR, KC, and K2B for both backbones. Both backbones preserve the contrast between broad retention and selective realization: Dynamic Cheatsheet has 100.0% KR but lower K2B than EvoSkill (87.3% versus 93.9% on GPT; 87.7% versus 93.9% on Qwen). The combined view also exposes within-method changes across backbones: ACE gains KP but loses KC on Qwen. This suggests that the backbone influences not only execution-time application but also the

artifact construction process itself, since the same evolution mechanism operating on different backbone outputs can produce artifacts with different precision–coverage profiles. Appendix L examines process–outcome associations and these paired backbone differences in detail.

Table 16: Final KP, KR, KC, and K2B on both backbones (%).

		GPT-5.6-Terra				Qwen3.8-Flash-Next			
Class	Method	KP	KR	KC	K2B	KP	KR	KC	K2B
Ctrl	No update	NA	0.0	0.0	NA	NA	0.0	0.0	NA
Ctrl	Trace retrieval	63.8	100.0	46.9	87.8	62.6	100.0	46.5	88.3
Ctrl	**Ctrl Avg (2)**	**63.8**	**50.0**	**23.4**	**87.8**	**62.6**	**50.0**	**23.3**	**88.3**
CN	Memento-NP	65.0	100.0	49.5	89.5	64.1	100.0	49.8	90.2
CN	A-Mem	87.3	100.0	51.0	88.1	88.2	100.0	51.5	88.8
CN	**CN Avg (2)**	**76.1**	**100.0**	**50.2**	**88.8**	**76.2**	**100.0**	**50.6**	**89.5**
KB	ReasoningBank	96.8	100.0	53.3	91.1	96.9	100.0	53.7	91.6
KB	MemP-online	82.8	45.3	23.2	89.8	82.5	43.3	22.6	90.2
KB	ReMe	96.7	53.2	30.5	92.0	96.6	53.4	30.5	92.1
KB	**KB Avg (3)**	**92.1**	**66.2**	**35.7**	**91.0**	**92.0**	**65.6**	**35.6**	**91.3**
GW	ExpeL	94.7	100.0	54.4	91.2	94.3	100.0	55.4	91.5
GW	D. Cheatsheet	90.8	100.0	49.4	87.3	90.6	100.0	50.5	87.7
GW	ACE	93.8	71.3	42.5	93.6	94.8	71.7	42.1	94.1
GW	AWM-online	93.4	56.2	31.1	92.3	93.0	57.0	31.3	92.7
GW	**GW Avg (4)**	**93.2**	**81.9**	**44.3**	**91.1**	**93.2**	**82.2**	**44.8**	**91.5**
SL	EvoSkill	99.8	24.7	13.5	93.9	99.6	23.6	13.0	93.9
SL	Memento-Skills	99.1	37.4	20.9	93.3	98.6	35.8	20.2	93.9
SL	**SL Avg (2)**	**99.4**	**31.1**	**17.2**	**93.6**	**99.1**	**29.7**	**16.6**	**93.9**
All	**Evolving Avg (11)**	**90.9**	**71.7**	**38.1**	**91.1**	**90.8**	**71.3**	**38.2**	**91.5**
All	**Overall Avg (13)**	**88.7**	**68.3**	**35.9**	**90.8**	**88.5**	**68.1**	**35.9**	**91.2**

L Extended Evidence-Based Diagnostic Analysis

The main-text heatmap localizes missing evidence among failed executions. This section examines whether process metrics track execution outcomes and how their profiles change across backbones. We then separate reference-state composition from realization, compare unconditional and failure-conditioned prevalence, and link the resulting diagnoses to the intervention tests in Appendix M. Descriptive associations motivate targeted interventions rather than establish causal bottlenecks.

L.1 Process–Outcome Associations

Figure 18 relates final KP, KR, KC, and K2B to final FSR and AMR across eleven evolving methods, separately for each backbone. K2B has the strongest association with final FSR ($r = 0.890/0.896$ for GPT/Qwen) and AMR ($0.903/0.896$); rank correlations agree ($\rho = 0.855$–0.900). Leave-one-out K2B–FSR ranges of 0.853–0.921 and 0.856–0.926 confirm that no single method drives the positive association. Negative KR/KC coefficients reflect confounded method designs and shared execution evidence between K2B and the outcome metrics, rather than indicating that retention harms execution. The pattern motivates inspecting retained-but-unrealized requirements, but cannot isolate a causal intervention target by itself. The strong K2B correlation suggests a testable hypothesis: interventions that improve the realization of already-retained requirements should yield larger execution gains than interventions that merely expand retention without improving realization. This hypothesis is examined through targeted interventions in Appendix M.

L.2 Backbone Sensitivity of Artifact Construction and Use

Figure 19 displays within-method Qwen-minus-GPT changes in final artifact metrics from Table 16. Across evolving methods, K2B rises by 0.44 pp, whereas KP falls by 0.08 pp and KR by 0.31 pp; KC rises by only 0.13 pp. ACE gains KP (+1.02 pp) while losing KC (-0.32 pp), and Memento-Skills gains K2B (+0.63 pp) despite lower KR/KC ($-1.57/-0.65$ pp). Better realization can thus coexist with narrower retention. Cross-backbone method ranks remain similar ($\rho = 0.973$–1.000

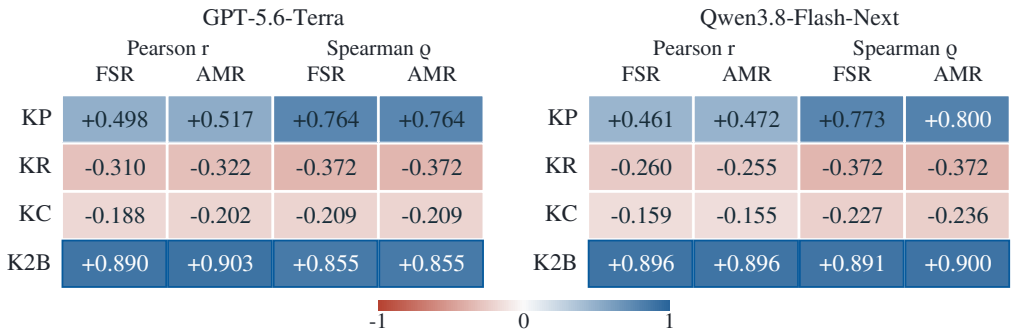


Figure 18: Process–outcome associations across eleven evolving methods per backbone.

across the four metrics), consistent with stable artifact profiles. The divergent metric shifts suggest that different backbones prioritize different trade-offs during artifact construction: Qwen may favor high-precision distillation at the cost of coverage, whereas GPT may retain broader content at the cost of realization accuracy. A separate writer–executor swap with fixed experience traces would be needed to attribute these differences specifically to artifact construction versus execution.

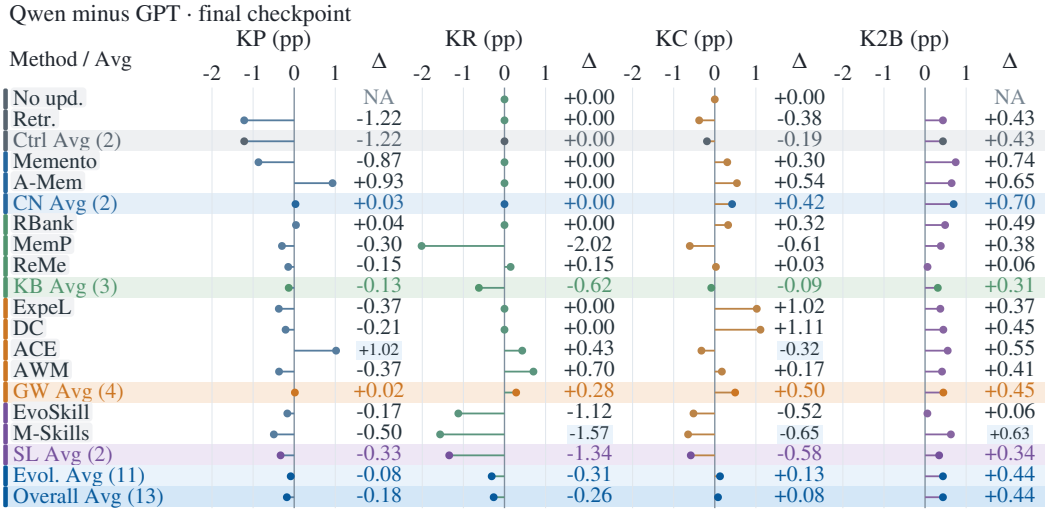


Figure 19: Within-method backbone differences in final artifact metrics (Qwen minus GPT, pp).

L.3 Reference-State Composition and Realization

Figure 20 partitions all reference requirements by experience encounter (E) and artifact retention (A), and compares realization rates in the encountered states. On GPT, case memories retain all encountered requirements but realize 85.9% of them, whereas skill libraries leave 37.7% of reference mass encountered but unretained yet realize 92.2% of the encountered-and-retained requirements. Skills also realize 86.8% of encountered-but-unretained requirements, which is close to the case-memory realization rate despite the absence of an explicit artifact. This suggests that the backbone's latent procedural capability, rather than the artifact alone, accounts for a substantial fraction of correct execution. Artifact absence therefore does not isolate an intervention target; these comparisons involve different reference subsets and are descriptive, not causal. A controlled experiment holding the reference subset fixed while varying only artifact availability would be needed to quantify the marginal contribution of retention.

Local Coverage–Execution Contrasts. The same distinction appears within individual families: GPT ReMe improves FSR from 86.5% to 93.8% on SWE-07 over 75–100% experience while KC

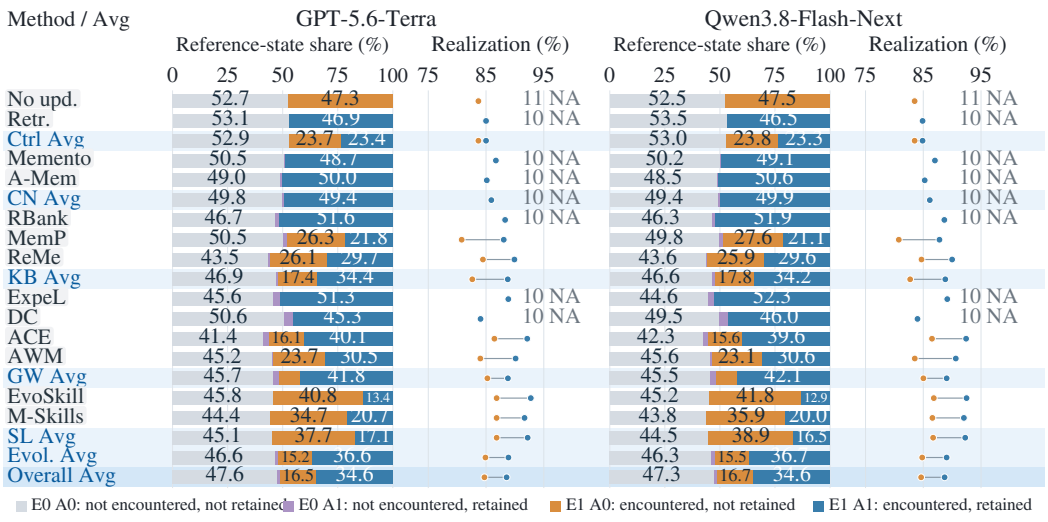


Figure 20: Reference-state composition and conditional realization on both backbones.

stays at 57.1%; EvoSkill raises KC from 21.1% to 26.5% on AFTER-11 over 50–75% experience while FSR stays at 58.3%. In the first case, execution improvement arises from better realization of the already-retained subset rather than from expanding coverage; in the second, the newly retained requirements are either inapplicable to the held-out instances or not effectively retrieved during execution. These dissociations confirm that artifact metrics and execution gains can diverge and that their correspondence requires examining what is retained against what is realized.

L.4 Failure Burden and Requirement-Normalized Violations

Figure 21 reports applicable requirement counts, miss prevalence among all and failed executions, and requirement-normalized violation rates. Operations are more numerous than decisions (median 6 versus 2) and affect more failed executions, yet Qwen Overall Avg violations are lower for operations (8.0%) than for decisions (15.0%) or dependencies (13.5%). The widest failure footprint thus differs from the highest normalized violation rate. This dissociation arises because operations are both more numerous and more likely to be satisfied: their higher absolute miss counts reflect their prevalence rather than their difficulty. Decisions and dependencies, though less numerous, pose harder challenges per applicable instance. Reducing operation misses may therefore improve more individual executions due to their prevalence, whereas reducing decision or dependency misses may yield larger per-requirement improvements.

M From Diagnosis to Targeted Interventions

This section evaluates whether the diagnostic profiles identify actionable targets. We apply two lightweight, stage-targeted modifications to the original frozen final state. For retention, three-model voting selects candidate knowledge for the persistent artifact. For execution-time reuse, an independent applicability review selects from the top-3 retrieved knowledge before it is supplied for execution. We compare both modifications with the original state and examine their limits. Evidence-processing validity is evaluated separately in Appendix P.

M.1 Stage-Targeted Interventions versus Original State

Each intervention is applied to the original frozen final state while leaving the remaining method state unchanged. The retention intervention uses three-model voting to select candidate knowledge for the persistent artifact. The retrieval intervention adds an independent applicability review over the top-3 retrieved knowledge before execution.

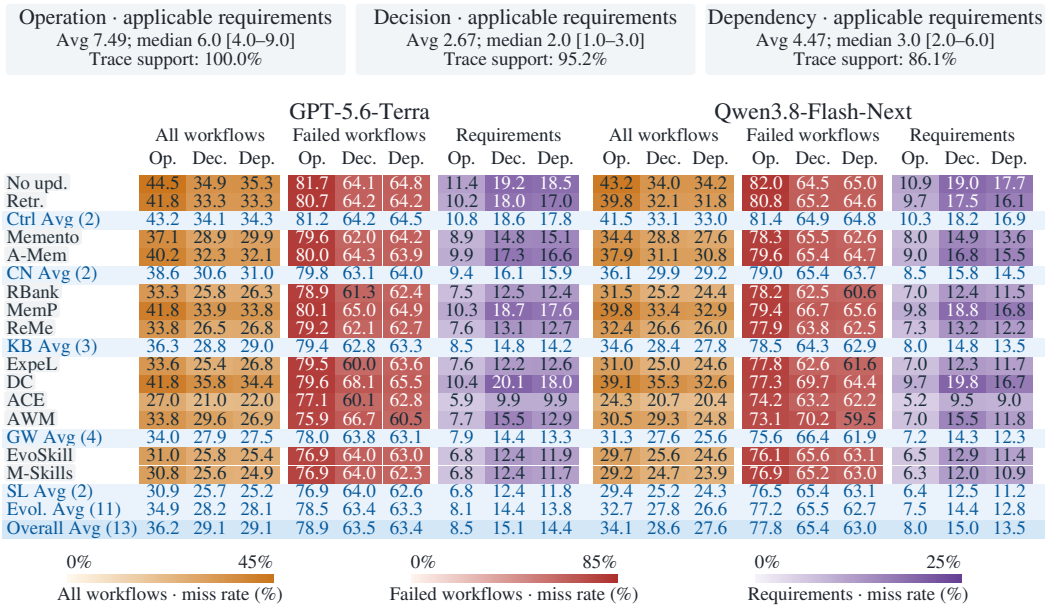


Figure 21: Requirement counts, miss prevalence, and normalized violation rates (%).

Table 17 reports stage-specific evaluation support and original-state scores. All held-out instances in an eligible cell remain in evaluation. The retention and retrieval evaluation subsets can overlap, so their effects cannot be summed. The support remains fixed between original and post-intervention states. KC pools retained/reference counts; FSR is the complete execution success fraction, while AMR and K2B average eligible trace fractions equally.

Table 17: Stage-specific evaluation support and original-state scores (%).

Stage	Backbone	Methods	Families	Cells	Tests	FSR	Process
Retention	GPT	6	60	845	18,464	52.81	28.49
Retention	Qwen	6	60	857	18,700	54.68	27.30
Retrieval	GPT	7	42	570	12,456	40.67	89.94
Retrieval	Qwen	7	44	554	12,144	42.77	90.25

Figure 22 shows changes in process measures and complete execution success relative to each evaluation subset's original state. The retention intervention gains 6.31 pp FSR on GPT and 6.33 pp on Qwen; the retrieval intervention gains 5.85 and 6.04 pp, respectively. Both modifications improve execution, but the similar magnitude reflects different underlying mechanisms: the retention intervention addresses missing artifact content, whereas the retrieval intervention addresses execution-time selection.

M.2 Trace-Guided Intervention across Methods

Figure 23 compares trace-guided intervention with the original final state for every evolving method and both backbones. Overall FSR rises by 5.43 pp on GPT and 5.37 pp on Qwen. Method effects differ in magnitude, and larger execution gains do not consistently correspond to larger requirement-level gains. These gains follow acquisition and are distinct from the evolution gains in Table 2.

M.3 Sensitivity and Limits of Intervention

Figure 24 separates the effects of reduced application rate and reduced intervention quality on a fixed evaluation subset from the full-subset outcomes. On GPT, 6.26% of executions are corrected and 0.83% are degraded, yielding +5.43 pp net FSR; Qwen yields +5.37 pp. Lower application rate and

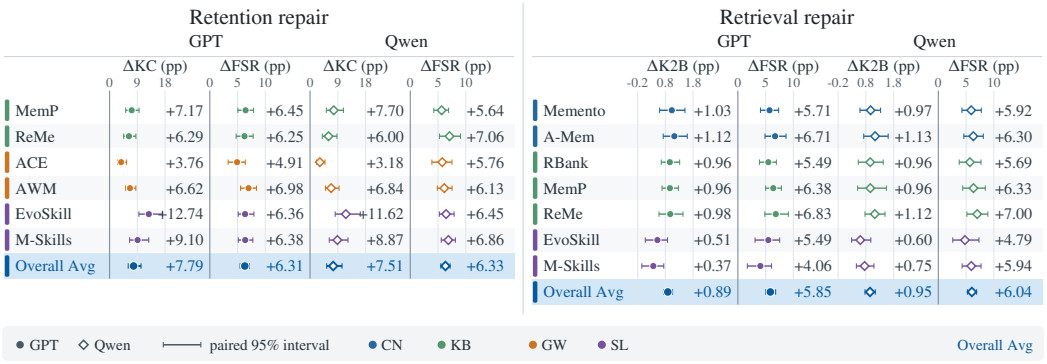


Figure 22: Method-level intervention-minus-original changes for the retention and execution-time reuse modifications (pp).

intervention quality attenuate the gains, and a local process-metric improvement does not guarantee that the complete execution succeeds. Remaining failures persist even when the targeted process metric improves.

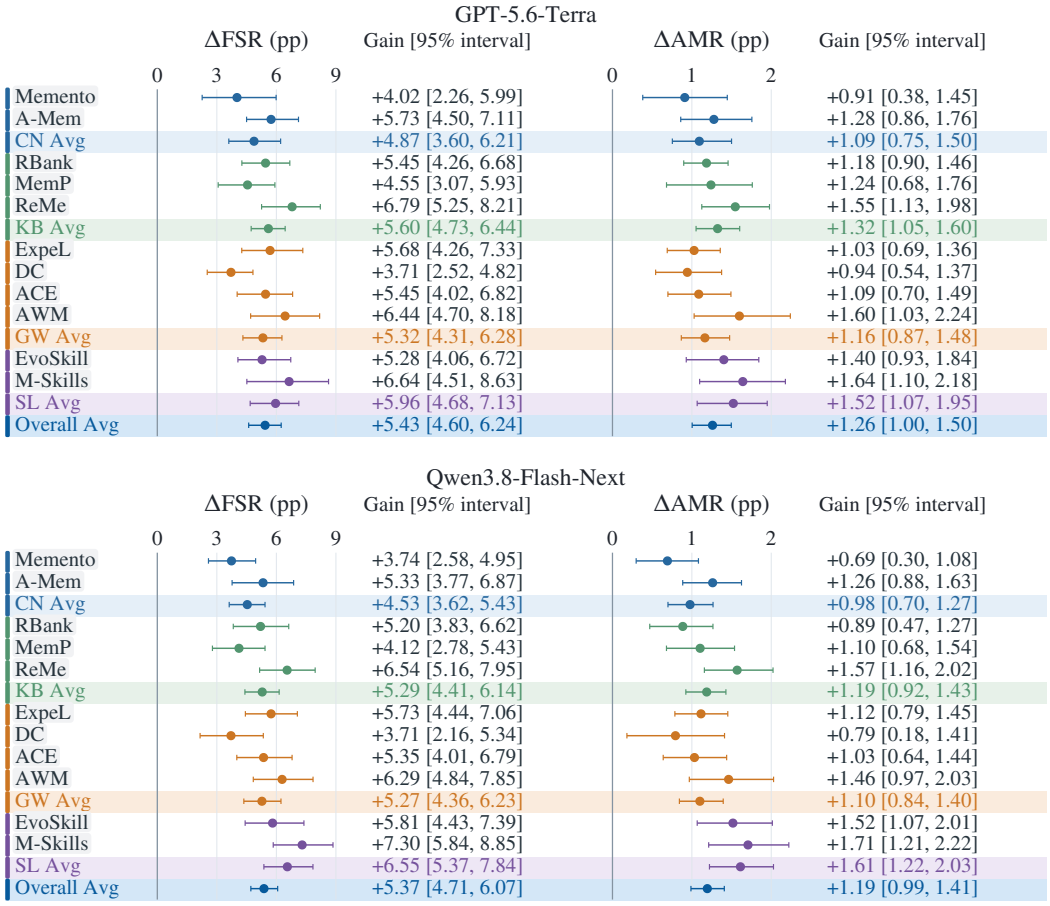


Figure 23: Intervention-minus-original FSR/AMR gains across all evolving methods (pp).

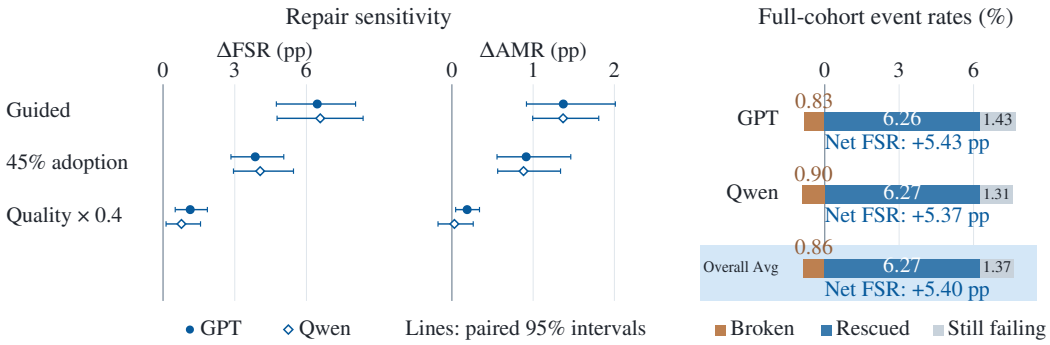


Figure 24: Intervention sensitivity and full-subset outcome rates relative to the original state (pp).

N Robustness and Sensitivity Analyses

We examine how final performance and learning gains vary with procedural source, procedure complexity, and backbone configuration on the evaluated families. These comparisons describe robustness within the evaluated procedures; they do not establish transfer to unseen sources or unseen procedural structures.

N.1 Results by Procedural Source

Figure 25 reports final FSR and AMR across the six procedural sources on both backbones. Source differences affect execution success more strongly than requirement completion: GPT ACE reaches 41.4% FSR on SOPBench versus 81.9% on SkillFlow, while AMR is 92.8% versus 93.9%. Dynamic Cheatsheet trails trace retrieval on SOPBench FSR on both backbones (23.1% versus 25.3%; 27.2% versus 30.6%). Because source labels combine differences in procedural complexity, domain conventions, and experience composition, these contrasts reflect the joint effect of these factors rather than an isolated difficulty dimension. The wide per-source variation nevertheless underscores that aggregate gains can obscure domain-specific weaknesses: a method that performs well overall may still underperform even a non-evolving baseline on specific procedural sources.

N.2 Performance by Procedure Complexity

Figure 26 reports final FSR/AMR and from-initial gains by reference requirement count on both backbones, splitting the 60 families into Low (14–36 requirements, 20 families), Middle (37–63, 21 families), and High (68–587, 19 families) bins. Performance decreases with procedural complexity: GPT Evolving Avg FSR falls from 74.6% (Low) to 56.4% (Middle) and 36.3% (High), while AMR contracts less sharply (91.1% versus 89.3% versus 89.0%), consistent with the general pattern that partial requirement completion is more resilient than complete execution success. Gains exhibit a similar gradient: Evolving Avg FSR gain is +12.2 pp (Low) and +6.7 pp (High) on GPT; Qwen follows the same ordering (+14.0 versus +9.6 pp). The decline is not uniform across methods: ACE maintains a gain of +19.7 pp in the High bin on Qwen, whereas Dynamic Cheatsheet drops to +2.2 pp, suggesting that multi-role evolution degrades more gracefully than monolithic document accumulation as the reference set grows. Cross-backbone differences also interact with complexity: Qwen exceeds GPT in Evolving Avg FSR gain by 1.8 pp in the Low bin and by 2.9 pp in the High bin, indicating that the stronger backbone's advantage widens rather than narrows as procedural complexity increases. Because bins are defined by total reference requirement count—which confounds branching depth, requirement heterogeneity, and domain difficulty—these contrasts describe complexity-bin sensitivity rather than the isolated effect of any single structural dimension.

N.3 Additional Backbone Configurations

Figure 27 compares from-initial FSR/AMR gains across three Qwen configurations: Qwen3.8-27B (dense) (Qwen Team, 2026a), Qwen3.8-Flash-Next (Qwen Team, 2026b), and Qwen3.8-2.4T-A95B (2.4T, 95B active) (Qwen Team, 2026c), for six selected conditions on the same 60 families and

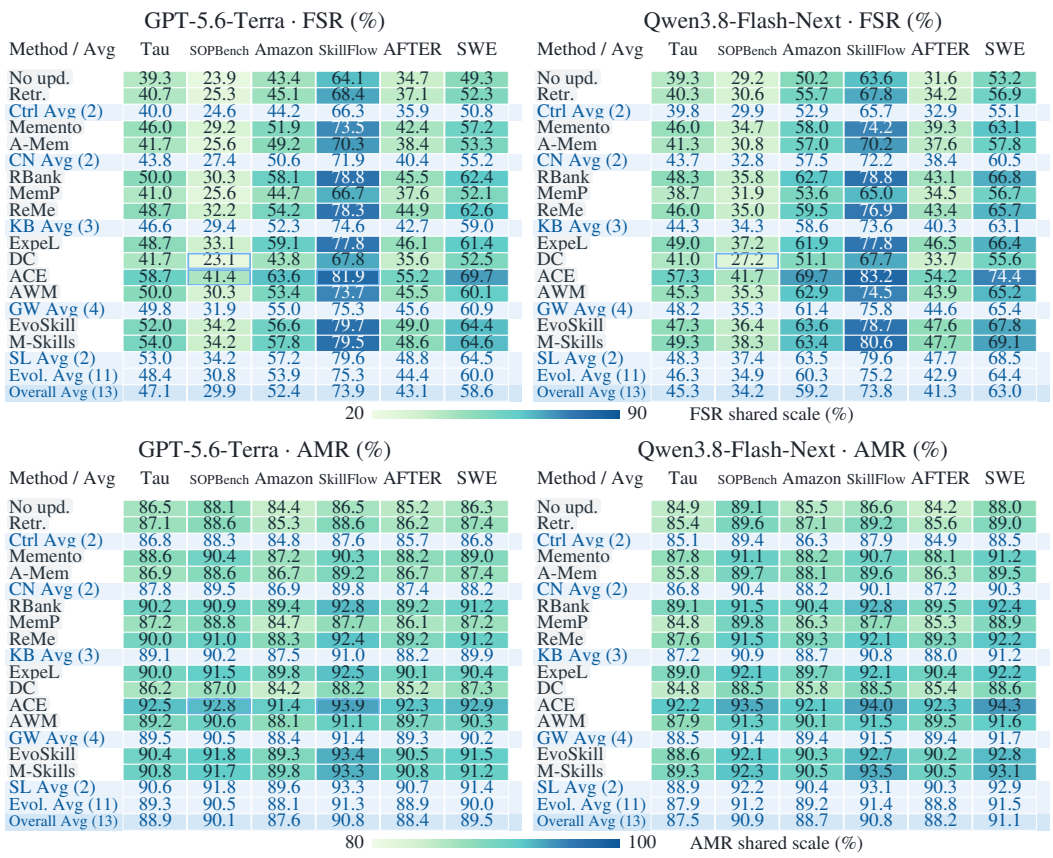


Figure 25: Final FSR and AMR by procedural source on both backbones (%).

three experience orders. Flash-Next results reuse the main-text evaluation data; the dense and MoE configurations are evaluated under the same protocol but do not isolate parameter count from other architectural differences. Learning gains remain positive for the four selected evolving methods across all three configurations, but higher initial capability does not imply larger gains: Evolving Avg FSR increases by 14.67, 15.47, and 12.22 pp, respectively. The largest gains occur at the intermediate Flash-Next tier rather than at either extreme, suggesting that evolution effectiveness depends on a capability sweet spot rather than monotonically increasing with model scale. Metric-specific reversals also persist: Flash-Next exceeds 27B in ACE's FSR gain, while 27B has the larger AMR gain. These configuration-specific interactions indicate that conclusions drawn from a single backbone may not generalize across the capability spectrum.

N.4 Sensitivity across Experience-Budget Strata

Figure 28 compares from-initial FSR/AMR gains in the short- and long-budget family strata on both backbones. The strata contain 25 and 17 families, respectively; the remaining 18 families fall outside this contrast. Evolving Avg FSR gains are larger in the long-budget stratum: 11.52 versus 5.44 pp on GPT and 12.66 versus 10.43 pp on Qwen. The smaller Qwen gap indicates that this pattern is not uniform across backbones, suggesting that backbone capability interacts with experience volume: a more capable backbone may extract more from limited experience, narrowing the advantage of additional budget. Because family composition also changes between strata, these contrasts describe budget-stratum sensitivity rather than the causal benefit of giving the same procedures more experience.

GPT-5.6-Terra

	Final FSR (%)			Final AMR (%)		
	Low	Middle	High	Low	Middle	High
No update	64.0	45.5	27.5	86.7	85.5	85.9
Retrieval	67.4	48.3	29.3	88.5	86.5	86.6
Memento-NP	72.4	54.1	34.2	90.1	88.4	88.4
A-Mem	68.4	50.8	30.6	88.5	87.4	86.9
ReasoningBank	78.4	58.9	36.9	92.6	90.4	89.4
MemP-online	66.4	47.4	30.3	87.8	86.1	86.8
ReMe	78.2	57.5	36.7	92.4	90.0	89.4
ExpeL	76.2	59.8	37.9	91.8	90.5	90.0
D. Cheatsheet	68.1	46.8	28.2	88.0	85.9	85.8
ACE	81.5	66.9	47.1	93.7	92.6	92.0
AWM-online	73.5	56.6	37.0	91.0	89.6	89.4
EvoSkill	79.2	60.5	39.8	93.1	90.7	90.2
M-Skills	79.0	61.3	40.0	92.9	91.0	90.3
Evolving Avg (11)	**74.6**	**56.4**	**36.3**	**91.1**	**89.3**	**89.0**

Qwen3.8-Flash-Next

	Final FSR (%)			Final AMR (%)		
	Low	Middle	High	Low	Middle	High
No update	64.4	47.7	30.3	86.7	86.6	86.2
Retrieval	68.5	52.5	31.7	88.9	87.8	86.9
Memento-NP	74.1	57.4	37.0	91.1	89.4	88.9
A-Mem	70.1	53.8	33.9	89.5	88.3	87.5
ReasoningBank	78.4	62.7	38.5	92.8	91.2	90.0
MemP-online	66.4	51.8	31.9	87.9	87.5	86.7
ReMe	76.8	60.4	38.4	92.2	90.6	89.6
ExpeL	76.9	63.1	40.9	92.0	91.2	90.5
D. Cheatsheet	68.2	49.5	30.9	88.4	86.8	86.6
ACE	83.7	70.0	48.3	94.7	92.8	92.5
AWM-online	74.6	62.1	38.5	91.4	90.9	89.7
EvoSkill	79.1	63.6	40.8	93.0	91.3	90.4
M-Skills	80.6	64.3	41.8	93.8	91.7	90.5
Evolving Avg (11)	**75.3**	**59.9**	**38.3**	**91.5**	**90.1**	**89.4**

GPT-5.6-Terra

	Gain FSR (pp)			Gain AMR (pp)		
	Low	Middle	High	Low	Middle	High
No update	+1.6	-3.4	-2.0	+0.1	-0.5	-0.0
Retrieval	+5.0	-0.5	-0.2	+1.9	+0.6	+0.7
Memento-NP	+10.0	+5.2	+4.7	+3.4	+2.4	+2.5
A-Mem	+6.0	+2.0	+1.1	+1.9	+1.5	+1.0
ReasoningBank	+16.0	+10.0	+7.4	+6.0	+4.4	+3.5
MemP-online	+3.9	-1.4	+0.8	+1.2	+0.1	+0.9
ReMe	+15.7	+8.6	+7.2	+5.7	+4.0	+3.5
ExpeL	+13.7	+10.9	+8.4	+5.2	+4.5	+4.1
D. Cheatsheet	+5.6	-2.0	-1.3	+1.4	-0.0	-0.1
ACE	+19.1	+18.0	+17.6	+7.1	+6.7	+6.1
AWM-online	+11.0	+7.7	+7.5	+4.3	+3.6	+3.5
EvoSkill	+16.7	+11.6	+10.3	+6.5	+4.7	+4.3
M-Skills	+16.6	+12.4	+10.5	+6.2	+5.1	+4.4
Evolving Avg (11)	**+12.2**	**+7.5**	**+6.7**	**+4.4**	**+3.4**	**+3.1**

Qwen3.8-Flash-Next

	Gain FSR (pp)			Gain AMR (pp)		
	Low	Middle	High	Low	Middle	High
No update	+3.1	+0.0	+1.6	+0.9	+0.9	-0.1
Retrieval	+7.2	+4.7	+3.0	+3.1	+2.1	+0.6
Memento-NP	+12.7	+9.7	+8.3	+5.3	+3.7	+2.7
A-Mem	+8.7	+6.1	+5.3	+3.7	+2.6	+1.2
ReasoningBank	+17.1	+14.9	+9.8	+7.0	+5.5	+3.7
MemP-online	+5.1	+4.1	+3.2	+2.0	+1.8	+0.4
ReMe	+15.4	+12.7	+9.7	+6.3	+5.0	+3.3
ExpeL	+15.5	+15.3	+12.2	+6.1	+5.5	+4.3
D. Cheatsheet	+6.9	+1.7	+2.2	+2.6	+1.1	+0.4
ACE	+22.4	+22.3	+19.7	+8.9	+7.1	+6.2
AWM-online	+13.3	+14.3	+9.8	+5.6	+5.2	+3.4
EvoSkill	+17.7	+15.8	+12.1	+7.2	+5.6	+4.1
M-Skills	+19.2	+16.6	+13.1	+8.0	+6.0	+4.2
Evolving Avg (11)	**+14.0**	**+12.1**	**+9.6**	**+5.7**	**+4.5**	**+3.1**

Figure 26: Final FSR/AMR (top, %) and own-initial gains (bottom, pp) by reference requirement count.

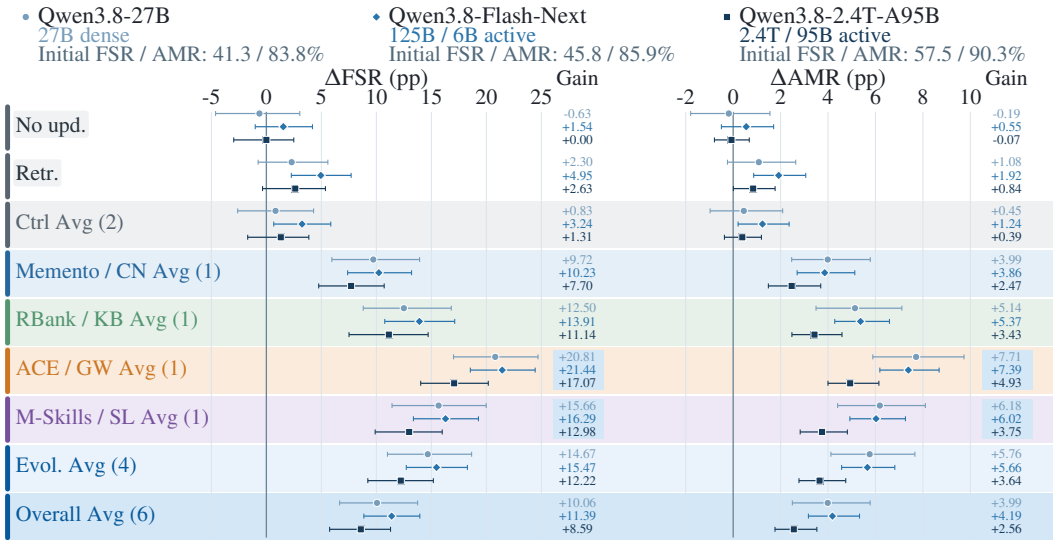


Figure 27: Learning gains across three Qwen configurations (pp).

Method / Avg	GPT-5.6-Terra				Qwen3.8-Flash-Next			
	ΔFSR 8-18 instances	ΔFSR 36-72 instances	ΔAMR 8-18 instances	ΔAMR 36-72 instances	ΔFSR 8-18 instances	ΔFSR 36-72 instances	ΔAMR 8-18 instances	ΔAMR 36-72 instances
No upd.	-2.8	-1.0	+0.0	-0.9	+2.2	+0.2	+1.4	-0.9
Retr.	-1.0	+3.2	+0.9	+1.0	+4.8	+4.8	+2.3	+1.3
Ctrl Avg (2)	-1.9	+1.1	+0.5	+0.1	+3.5	+2.5	+1.9	+0.2
Memento	+4.0	+9.0	+2.6	+2.7	+9.6	+10.0	+3.7	+3.8
A-Mem	+1.3	+3.8	+1.5	+1.1	+6.8	+7.4	+3.0	+2.2
CN Avg (2)	+2.7	+6.4	+2.1	+1.9	+8.2	+8.7	+3.4	+3.0
RBank	+7.5	+14.6	+3.9	+5.4	+12.8	+13.9	+5.1	+5.6
MemP	+0.1	+1.7	+1.0	+0.1	+4.3	+3.2	+2.3	+0.4
ReMe	+6.8	+12.9	+3.7	+4.7	+10.8	+13.4	+4.5	+5.1
KB Avg (3)	+4.8	+9.7	+2.9	+3.4	+9.3	+10.1	+4.0	+3.7
ExpeL	+8.1	+12.7	+4.1	+5.1	+13.3	+15.0	+5.2	+5.6
DC	-0.8	+1.8	+0.7	-0.2	+5.0	+2.8	+2.3	+0.7
ACE	+14.0	+21.1	+5.5	+7.5	+19.6	+22.4	+6.9	+7.9
AWM	+5.5	+10.8	+3.3	+4.1	+11.1	+12.1	+4.8	+4.3
GW Avg (4)	+6.7	+11.6	+3.4	+4.1	+12.3	+13.1	+4.8	+4.6
EvoSkill	+6.6	+19.4	+3.4	+7.2	+10.6	+18.8	+4.4	+6.9
M-Skills	+6.7	+19.0	+3.4	+6.8	+10.8	+20.3	+4.6	+7.4
SL Avg (2)	+6.7	+19.2	+3.4	+7.0	+10.7	+19.6	+4.5	+7.2
Evol. Avg (11)	+5.4	+11.5	+3.0	+4.1	+10.4	+12.7	+4.3	+4.5
Overall Avg (13)	+4.3	+9.9	+2.6	+3.4	+9.4	+11.1	+3.9	+3.9

Shared color scales: ΔFSR −35 to +35 pp; ΔAMR −10 to +10 pp. Zero is white.

Figure 28: From-initial FSR/AMR gains across short- and long-budget family strata (pp).

N.5 Experience-Order Sensitivity

Figure 29 reports from-initial FSR/AMR gains and the gain difference against trace retrieval under the same order, using the common held-out instances across all three shared experience orders. Nine of eleven evolving methods exceed retrieval's FSR gain under all three orders on each backbone. Positive from-initial gains alone are insufficient to establish improvement over retrieval: Qwen Dynamic Cheatsheet gains 2.95–4.02 pp yet trails matched retrieval by 0.83–2.05 pp, indicating that its absolute improvement is attributable to shared experience exposure rather than effective evolution. GPT has four negative method/order FSR gains, all from MemP-online and Dynamic Cheatsheet; Qwen has none. Pairing with retrieval under the same order also changes the variability comparison: GPT ACE spans 15.76–21.74 pp in from-initial gain but only 16.06–17.27 pp against retrieval, because much of the raw variation is shared with the baseline through common instance ordering. Three runs describe observed stability without isolating a causal order effect or establishing a universal ranking.

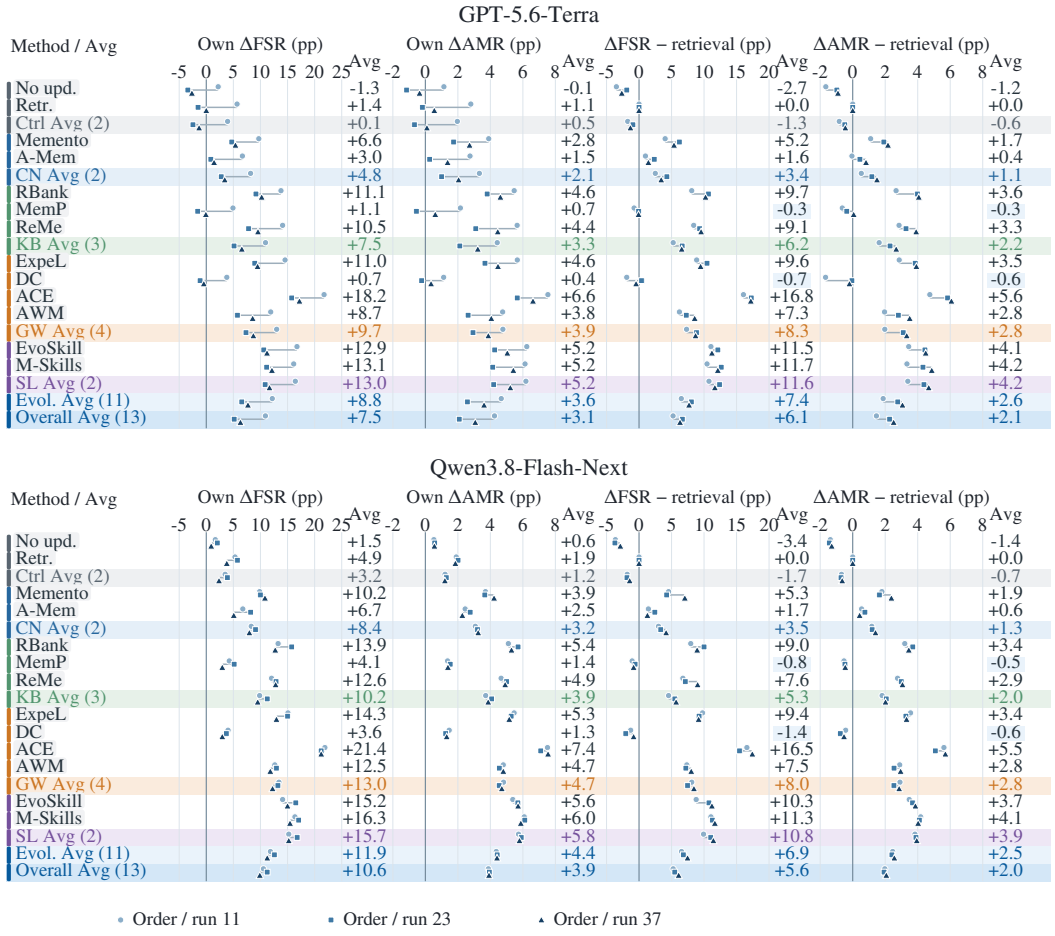


Figure 29: Paired experience-order sensitivity on both backbones (pp).

N.6 Cross-SOP Experience Mixing

Figure 30 extends the family-isolated analysis with shared-state evaluations under controlled mixing schedules. Each shared-state condition is run from reset state under the mixing schedule; only the matched isolated comparators reuse family-specific evaluation results as anchors. Six three-SOP bundles per overlap setting (three related/conflicting, three distinct; 36 bundles total) use the same 12 experiences per SOP and three paired seeds, yielding 216 shared-state runs per method. Panel A compares final metrics against matched isolated metrics under blocked, alternating, and shuffled inputs for related/conflicting versus distinct SOPs. Panel B separately learns one SOP first, then measures its FSR/KR/KC/K2B change after 24 experiences from the other two, without revisiting it. Figure 30 reports results for all six evaluated conditions (No update, Trace retrieval, Memento-NP, ReasoningBank, ACE, Memento-Skills).

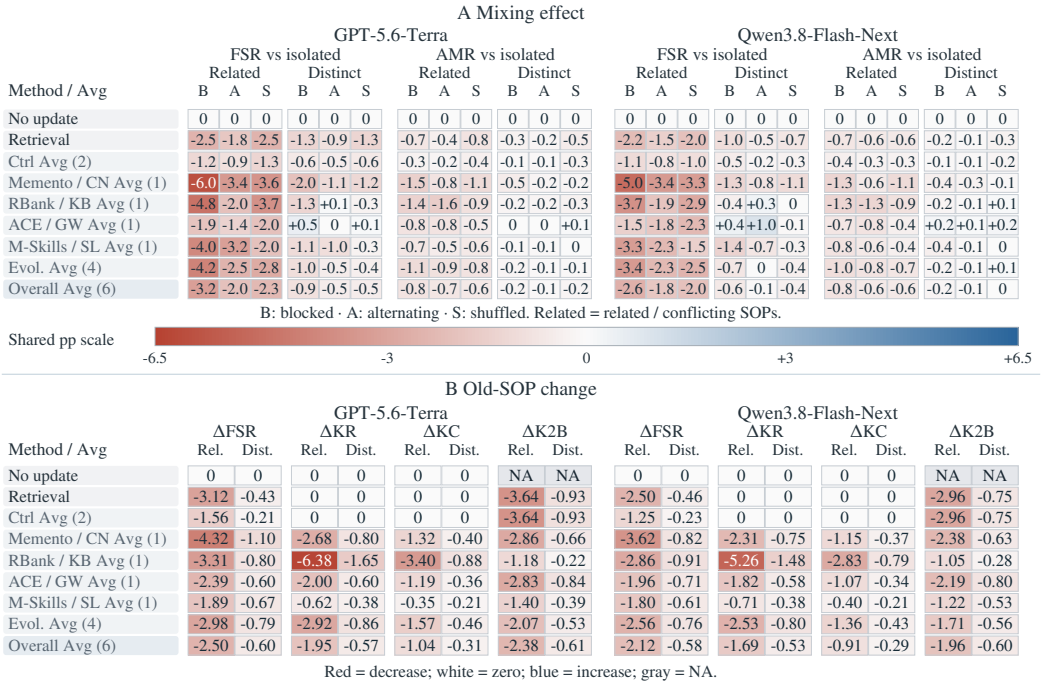


Figure 30: Cross-SOP mixing effects (A) and old-SOP retention changes (B), in pp.

The resulting patterns distinguish retention from use: under related-SOP interference, GPT retrieval keeps KR/KC unchanged while K2B and old-SOP FSR fall by 3.64 and 3.12 pp; ReasoningBank instead loses 6.38 pp KR and 3.40 pp KC. Schedule rankings also differ across methods and metrics. These cases illustrate that final gains measured in isolation cannot locate the source of interference: two methods may suffer comparable FSR degradation through entirely different pathways, one through artifact corruption and the other through retrieval confusion. Diagnosing cross-SOP effects therefore requires the same stage-level decomposition used for within-family analysis.

O Computational Cost and Resource Efficiency

Table 18 reports final resource accounting for both backbones. Task and evolution tokens are millions, elapsed time is hours, and state size is MB; all are totals over the 180 family/order runs. State sizes coincide across backbones and are shown once. Memento-Skills consumes 114.2M learning tokens on GPT, compared with 65.6M for ReasoningBank and 72.5M for ACE; its evolution cost alone is 66.1M. Trace retrieval has no evolution-model tokens but retains a 3,697.9 MB store. Evolution expense and persistent-state size thus expose different cost dimensions that experience counts alone omit: a method with modest token cost may still impose substantial storage overhead, and vice versa. In practice, the choice between evolution-heavy and storage-heavy methods depends on deployment constraints such as inference latency, memory budgets, and the frequency of artifact updates.

Table 18: Final experience and evolution costs across both backbones.

	GPT-5.6-Terra			Qwen3.8-Flash-Next			
Method	Task	Evol.	Time	Task	Evol.	Time	State
No update	48.1	0.0	105.7	52.9	0.0	116.3	0.0
Trace retrieval	95.2	0.0	209.3	104.7	0.0	230.3	3697.9
Ctrl Avg (2)	**71.6**	**0.0**	**157.5**	**78.8**	**0.0**	**173.3**	**1848.9**
Memento-NP	57.7	8.6	145.9	63.4	9.8	161.1	1848.9
A-Mem	48.1	14.2	137.1	52.9	16.1	151.7	2465.2
CN Avg (2)	**52.9**	**11.4**	**141.5**	**58.2**	**12.9**	**156.4**	**2157.1**
ReasoningBank	52.9	12.7	144.3	58.2	14.4	159.6	308.2
MemP-online	57.7	16.3	162.7	63.4	18.4	180.0	452.0
ReMe	48.1	30.5	172.9	52.9	34.5	192.2	402.7
KB Avg (3)	**52.9**	**19.8**	**160.0**	**58.2**	**22.4**	**177.3**	**387.6**
ExpeL	52.9	17.3	154.4	58.2	19.5	170.9	1232.6
D. Cheatsheet	57.7	21.4	173.9	63.4	24.1	192.7	1232.6
ACE	48.1	24.4	159.5	52.9	27.6	177.0	373.9
AWM-online	52.9	19.3	158.8	58.2	21.8	176.0	698.5
GW Avg (4)	**52.9**	**20.6**	**161.6**	**58.2**	**23.3**	**179.2**	**884.4**
EvoSkill	57.7	56.0	250.0	63.4	63.2	278.7	821.7
Memento-Skills	48.1	66.1	251.2	52.9	74.7	280.7	1027.2
SL Avg (2)	**52.9**	**61.1**	**250.6**	**58.2**	**69.0**	**279.7**	**924.5**
Evolving Avg (11)	**52.9**	**26.1**	**173.7**	**58.2**	**29.5**	**192.8**	**987.6**
Overall Avg (13)	**55.7**	**22.1**	**171.2**	**61.3**	**24.9**	**189.8**	**1120.1**

Figure 31 reindexes the existing FSR checkpoints by experience-plus-evolution tokens per run on both backbones. Higher endpoint performance can require a longer resource trajectory: skill-library methods reach competitive FSR but at roughly twice the token cost of knowledge-bank methods. These curves complement the instance-indexed dynamics by exposing the cost of reaching each observed checkpoint without extrapolating methods beyond their recorded budget.

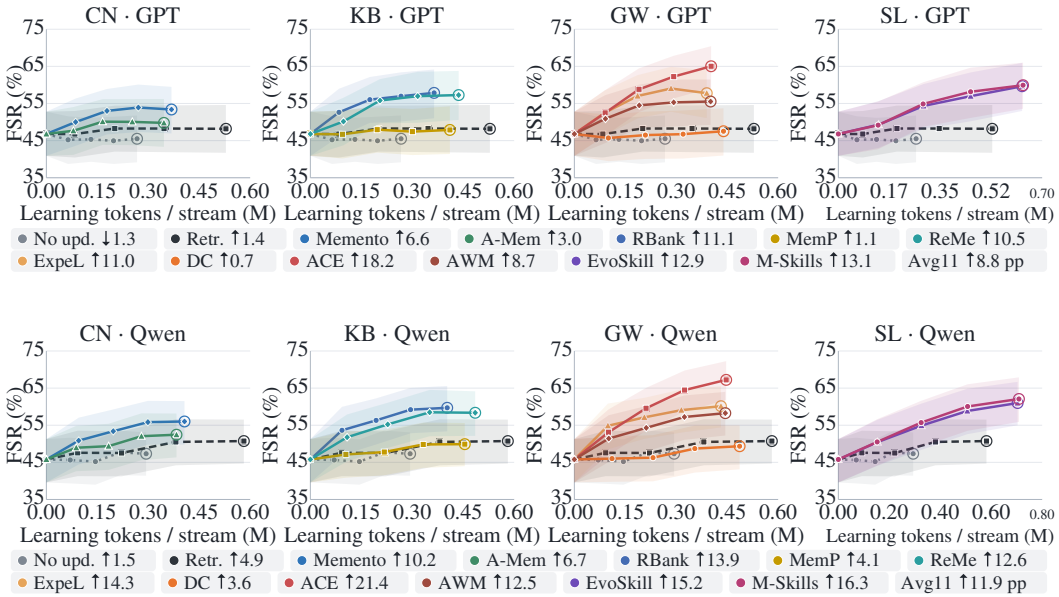


Figure 31: FSR versus cumulative learning tokens per run on both backbones.

P MEASUREMENT RELIABILITY AND DIAGNOSTIC VALIDITY

Table 19 separates extraction and binary judgment errors (A), their effects on metrics and diagnoses (B), and native versus pre-audit procedural outcomes (C). Panel B uses complete paired trajectories for FSR and trace-equal AMR; diagnostic changes use linked reference–episode records. Skills extraction recall is 74.2%, and decision and dependency agreement is 78.8% and 76.3%; aggregate agreement would obscure these difficulties. AMR corrections differ in sign across backbones, while the native–procedural disagreement rate is 20.5%. These contrasts motivate separate error and impact analyses.

Table 19: Measurement-reliability summary across extraction, judgment, and native-agreement dimensions.

A. Evidence reliability (2 independent reviewers + 1 adjudicator)

Artifact group	Units	Ref. claims	Auto claims	Precision	Recall
Cases	8	92	96	87.5	91.3
Knowledge	8	108	112	81.3	84.3
Guidance	8	116	111	88.3	84.5
Skills	8	124	108	85.2	74.2
Overall (pooled)	32	440	427	85.5	83.0

Separate absence check: 64 pairs; 58 resolved; false absence 15.5% of resolved; 6 unresolved.

Judgment channel	n	Agreement	False accept	False reject
Exposure support	240	87.5	13.0	12.2
Artifact matching	240	82.1	22.6	16.3
Operation execution	80	88.8	15.0	10.0
Decision execution	80	78.8	23.1	20.4
Dependency execution	80	76.3	34.5	17.6
Complete workflow	120	84.2	15.8	15.9

B. Consequences: same-trace metric and linked-diagnosis changes

Backbone	ΔFSR [interval]	ΔAMR [interval]	State changes	Retained-miss changes
GPT	+3.3 [-6.7, 13.3]	+0.5 [-2.9, 4.2]	18.8 [0.0, 37.5]	25.0 [6.3, 50.0]
Qwen	-1.7 [-11.7, 8.3]	-2.4 [-5.0, 0.2]	25.0 [6.3, 50.0]	18.8 [0.0, 37.5]

Per backbone: 60 trajectories, 16 linked records; 95% intervals from 10,000 paired-unit resamples.

C. Native versus pre-audit automatic FSR: 112 jointly evaluable trajectories

Native outcome	FSR success	FSR failure
Success	44.6	13.4
Failure	7.1	34.8

Native outcome unavailable: 6.7% of 120; disagreement: 20.5% of 112 jointly evaluable records.