SETA: OFFLINE-TO-ONLINE COMPUTE AMORTIZATION VIA EXPLICIT-FAILURE MEMORY FOR TOOL-USING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agents can fail a task and still judge themselves successful. Success-only memory discards this signal, and K-attempt self-consistency multiplies compute. SETA (Selective Explicit-failure Trajectory Amortization) turns externally verified failures into reusable memory. Offline, we collect four exploratory trajectories per task and flag false successes. We deterministically rewrite each: the error path stays, only the outcome changes, with no model call. At deployment, a single attempt retrieves this memory: a success skeleton, known-pitfall exemplars, and one explicit-failure memory. For example, a wrongly approved leave request becomes a note the agent reads when the same request returns.

We evaluate 60 requesting-time-off tasks and 50 customer-request-routing tasks from AgentArch, assuming mature memory for recurring requests; cold-start and cross-task transfer are out of scope. Under this assumption, a single Think-mode attempt reaches $\mathbf{85.00}\,\%$ final-outcome success on time-off tasks. On a separate seed set, removing only the explicit-failure memory from otherwise identical retrieval lowers final-outcome success by 16.00 points. A corrected bf16 harness reproduces this effect at 15.67 points; a second stack shows 9.00 points, and retrieval alone gives no gain on either. In Think mode, SETA exceeds a workflow-memory baseline, an LLM-written lesson-memory baseline, and four-attempt majority voting without memory. SETA and the workflow-memory baseline pay the same strict-task-success cost, but only SETA converts it into a gain; under non-thinking (NonThink) deployment, both are indistinguishable from no memory. On the longer routing tasks, memory injected once has no effect. Restating explicit-failure memory each call yields a 19.20-point gain; restating retrieval-only memory yields none.

1 INTRODUCTION

Agents can fail a task and still judge themselves successful. A standard accuracy lever for this problem is *K-attempt self-consistency / majority-vote decoding* (Wang et al., 2023); it improves accuracy but multiplies per-query compute. We instead study Think-mode (the backbone's extended-reasoning mode, switched on) offline exploration ($K = 4$) reused by a single-attempt ($K = 1$) online policy with same-task history. Same-task history means memory built from earlier offline attempts at the same recurring (repeated) requests, and the agent still re-executes every tool call rather than replaying a cached answer. This regime arises whenever an agent serves a few recurring workflow types, such as leave requests or case routing, and the same requests come back. If such runs are audited offline, verified failures of earlier attempts already exist, and the goal is that none of them is repeated, at single-attempt cost. Cross-task experience memories (Ouyang et al., 2025; Wang et al., 2024) instead aim to transfer lessons to new tasks; the two regimes are complementary, and we make no cross-task or unseen-family claim.

Open-weight agents accumulate trajectories, including "successful" traces that are actually wrong final answers: *false successes* (FS), also called *silent failures* (Anonymous, 2026). Because FS traces expose confident errors, we deterministically convert them to *explicit-failure (EF) memories* offline and retrieve them at single-attempt inference time. Throughout, final-outcome success means the agent's final answer is correct; strict task success also checks tool use (§4).

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.

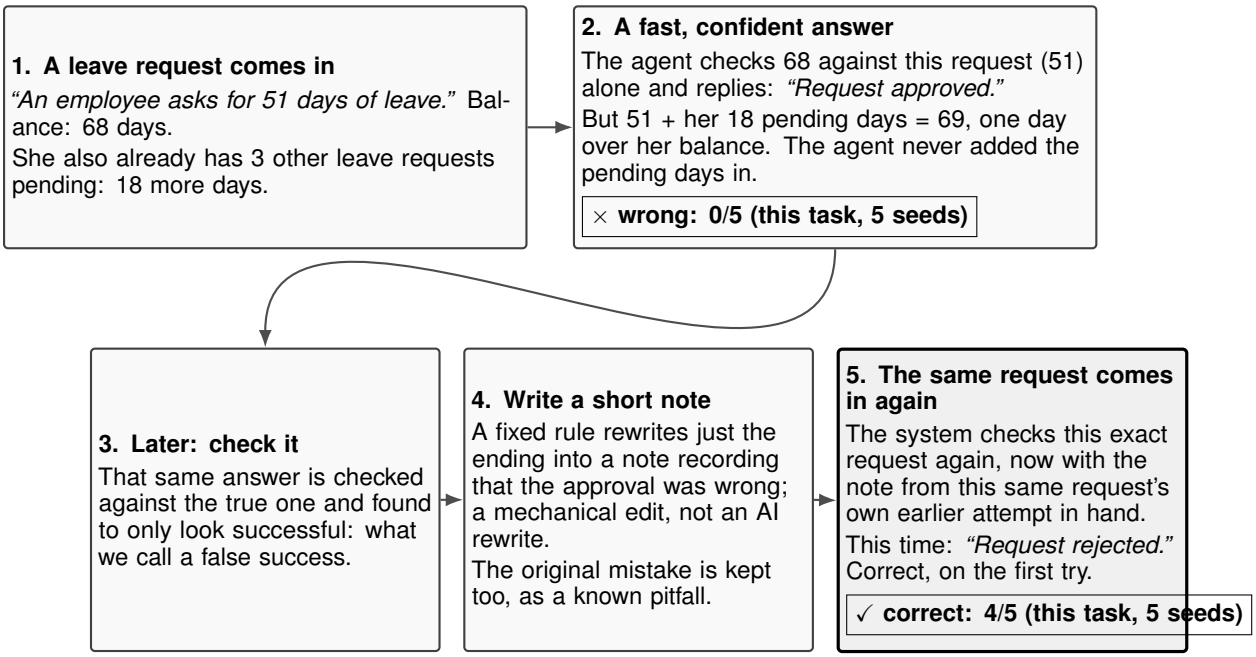


Figure 1: **One wrong answer, fixed for next time.** Without memory, the agent approves this leave request, missing that pending leave pushes the total one day over. A fixed rule (no model call) rewrites that failure into a note that the approval was wrong, and the note fixes the same request next time. Counts ($\times$ 0/5, ✓ 4/5) are one task's 5-seed pass rate, not the headline result.

Contributions. The contribution is methodological, scoped to the recurring-request deployment regime above. Two workflows serve distinct roles: short requesting-time-off (RTO) tasks isolate the mechanism, and longer customer-request-routing (CRR) tasks isolate memory placement.

1. **Auditable failure-memory construction.** We define a provenance-preserving deterministic operator g that converts selected false-success traces into explicit-failure memories by rewriting terminal fields while preserving the source trajectory; construction is single-shot and model-call-free. "Selective" in SETA's name refers to this selection step (which trajectories the judge routes to rewriting).
2. **The EF diagnosis carries the effect, and the effect reproduces.** We show the gain tracks the EF diagnosis, not retrieval exposure. Adding EF on top of retrieval-only gives $+16.00$ percentage points (pp) on the original stack (a fixed hardware, precision, and inference-engine version), $+15.67$ pp on the corrected bf16 stack, and $+9.00$ pp on the corrected FP8 stack; retrieval alone over no memory is not significant on either corrected stack ($+0.33$/$+1.00$ pp). The gain also requires task-appropriate content: content-mismatched controls are indistinguishable from retrieval-only, up to 8.33 pp below SETA (App. Y).
3. **Gains against baselines.** In Think mode, SETA exceeds a workflow-memory baseline (AWM) by $+11.67$ pp and an LLM-written lesson-memory baseline by $+8.33$ pp, and outperforms four-attempt majority voting without memory by $+10.00$ pp (§5.2).
4. **Delivery matters: placement, boundary conditions, and cost.** The effect requires Think-mode deployment; it is null for both SETA and AWM under NonThink (the same mode switched off; §5.3). On long (CRR) trajectories a single injection has no effect ($69.60\,\%$, tied with no memory), but restating memory every call reaches $88.80\,\%$ ($+19.20$ pp over no memory) and re-retrieving every call reaches $80.00\,\%$ ($+10.40$ pp); on short (RTO) trajectories, per-step retrieval ($84.33\,\%$) is no different from single-injection SETA (§5.4). A restated retrieval-only control confirms this gain needs the EF diagnosis: it reaches

only $68.40\,\%$, 20.40 pp below restated EF ($p < 0.001$; §5.4). The smaller FP8 effect traces to one serving stack, not to precision (difference-in-differences -0.33 pp, $p = 0.97$; App. S). SETA's cost in strict task success under Think mode is shared with AWM and the lesson-memory baseline, not unique to SETA.

Out of scope. SETA targets recurring-request deployment; we make no cross-task generalization or strict-cold-start accuracy-improvement claim.

2 Related Work

Tool agents and trajectory reuse. AgentBench (Liu et al., 2023), ToolBench (Qin et al., 2023), and τ-bench (Yao et al., 2024) evaluate tool use. Voyager (Wang et al., 2023), Reflexion (Shinn et al., 2023), and ExpeL (Zhao et al., 2023) reuse trajectories as exemplars, usually accepting success labels. SETA instead retains FS pitfalls and adds deterministic, LLM-call-free EF rewrites.

Reflection. Reflexion (Shinn et al., 2023) reflects within a trajectory; SETA edits false successes (FS) across trajectories, which within-trajectory reflection cannot detect. An uncontrolled five-seed sanity run on our RTO task set reaches $76.00\,\%$ final-outcome correctness at trial one and declines over later trials (not compute-matched; Appendix N). A controlled, payload-verified replication finds Reflexion's costliest stage indistinguishable from no memory and significantly below SETA, at several times SETA's cost (Appendix N.1).

RAG and memory. RAG (Lewis et al., 2020) retrieves evidence at inference time; SETA instead indexes failure-aware (EF) memory offline with Think-mode exploration ($K = 4$) and queries it online with a single attempt ($K = 1$). Agent Workflow Memory (AWM; Wang et al., 2024) induces reusable workflows from demonstrations; Mistake Notebook Learning (Su et al., 2025) clusters failures into structured notes; Memento (Zhou et al., 2025) learns case-based memory selection from feedback (Table 24, App. U). SETA is statistically indistinguishable from AWM under NonThink but ahead under Think mode (§5.3, App. S); Mistake Notebook Learning and Memento were not compute-matched. ReasoningBank (Ouyang et al., 2025), Dynamic Cheatsheet (Suzgun et al., 2025), and AutoGuide (Fu et al., 2024) build memory from successes and failures via a model call; A-MEM (Xu et al., 2025) and Agent KB (Tang et al., 2025) instead focus on memory linking and cross-framework sharing. SETA needs no model call and stays specific to the failing task, unlike all five (App. Y; survey: (Huang et al., 2026)).

Combined contribution. FS-only selection, deterministic provenance-preserving EF rewriting, offline-cost amortization via repeated reuse, and matched mechanism-isolation controls together form an auditable failure-aware memory construction and deployment protocol. **Test-time compute.** Scaling inference compute can improve quality (Snell et al., 2024; Brown et al., 2024); SETA is complementary, paying a one-time offline cost and reusing it rather than adding online compute. **Deployment evaluation.** We separate mature-memory reuse from strict cold-start.

LLM judges. We treat judges as instruments, not oracles; inter-judge agreement is high and reported in §5.1, with prompts and breakdown in Appendix A.

3 Method

Figure 2 shows the full pipeline. Offline, SETA turns confidently-wrong trajectories into explicit-failure memories with no model call; online, a single Think-mode attempt retrieves that memory, and on long workflows keeps seeing it at every orchestrator call.

Outcome taxonomy. Each (query, trajectory, outcome) is one of four classes, judged by whether the agent's final answer matches ground truth and whether its own internal detector agrees: **TS** (true success, both agree it succeeded), **EF** (explicit failure, both agree it failed), **FS** (false success, the agent is confidently wrong; SETA's rewrite target g), and **FF** (final failure, an incomplete or refused answer). A closed reference judge c_ϕ implements this taxonomy in our main experiments; App. C examines self-classifier substitution (full notation: Appendix A.6).

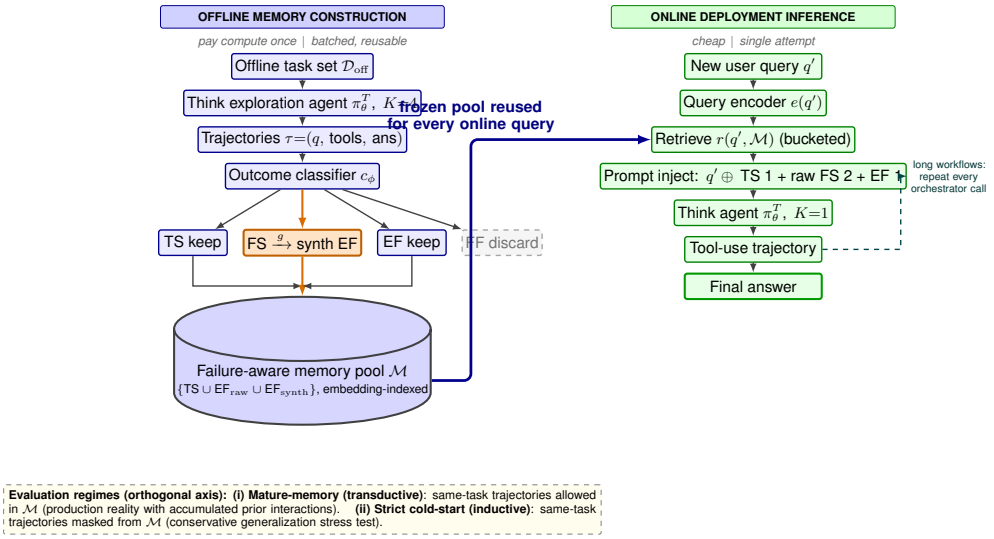


Figure 2: **SETA offline-to-online compute-amortization architecture.** Offline (left), SETA collects $K=4$ Think-mode exploration trajectories and classifies outcomes into TS / EF / FS / FF. It converts false successes (FS) by deterministic rewriting into explicit-failure (EF) memories via g (orange path), then indexes the resulting failure-aware pool $\mathcal{M}$: true successes, retained raw-FS pitfalls, and deterministic EF rewrites. The retriever selects 3/2/2 by bucket and the formatter renders 1/2/1 (four trajectories). Online (right), a $K=1$ Think agent retrieves from $\mathcal{M}$ and uses it as prompt context; on long, multi-call workflows the same block is restated or re-retrieved at every orchestrator call (dashed loop, §5.4), since a single injection is visible only at the first call. The pool is built once and reused per query, amortizing Think-mode offline exploration cost over deployment volume. Mature-memory vs. strict cold-start (inset) is an orthogonal axis on whether same-task trajectories are visible in $\mathcal{M}$.

Given a one-time offline compute budget and a tight per-query online budget, SETA builds a memory pool $\mathcal{M}$ once and defines an online policy

$$\pi_{\text{SETA}}(a \mid q) \triangleq \pi_\theta(a \mid q, r(q, \mathcal{M})),$$

where $r(q, \mathcal{M})$ retrieves the most relevant prior memory for q. The policy is defined independently of inference mode; π_θ runs configured Think-mode at online deployment for the headline and RTO retrieval-only/retrieval+EF measurements (original stack, seeds $\{42–46\}$), retrospectively supported by think-trace markers rather than request-payload verification (App. O). In mature-memory deployments, this is a single-attempt configuration compared against evaluated K-attempt references, not a claim of causal $K=4$ recovery at $K=1$ cost.

3.1 Offline Pool Construction

Offline, SETA builds a provenance-preserving failure-memory pool from four Think-mode exploratory trajectories per historical task ($K_{\text{offline}} = 4$), selecting false successes with the external judge c_ϕ. For each selected FS trace, g performs three steps: it deep-copies the trajectory, rewrites only the terminal outcome field with the audited diagnosis from c_ϕ, and records a provenance link back to the untouched raw FS trace, which remains available for inspection. Thus g is a deterministic, one-time rewrite with no model call. A 12-trial LLM re-prompt pilot produced zero valid FS→EF transitions; this negative method-selection audit is reported in Appendix A, not used to build the EF pool.

3.2 Online Retrieval and Inference

At deployment, one attempt receives four rendered memories from bucketed MMR (maximal marginal relevance): one TS skeleton, up to two raw-FS pitfalls, and one EF memory, so "top-1 retrieval" renders more than one memory (thresholds and surplus selections: Appendix A.6). $K=4$ controls use four real evaluation rollouts with majority aggregation; each $K=1$ headline uses one rollout per task.

3.3 Memory Placement in Long Trajectories

The policy above injects memory once, in the system prompt; multi-agent deployments call the orchestrator (the agent that coordinates the other sub-agents and tools) many times per task, so a single injection is not visible at later calls (Figure 2, dashed loop). Restated memory (SETA-R) re-appends the same retrieved block at every orchestrator call, adding prompt tokens but no new retrieval or model call. Per-step retrieval (SETA-S) instead re-retrieves at every call, from the same pool and filters, and the new block replaces the previous one. Appendix W gives the full design and cost (including the retrieval gate that routes multi-agent traces to this channel, Appendix A.6.4).

3.4 Validity of the Mature-Memory Regime

The mature-memory regime admits prior same-task trajectories into $\mathcal{M}$ and is **transductive by design** (scope limits: §1). It targets recurring requests. SETA still re-runs the tools on every attempt, because a recurring request can return different tool results (e.g., a changed leave balance); the memory guides the agent's decision rather than replacing it.

What the memory contains, and why the gain is not answer copying. An EF memory records the audited outcome of an earlier attempt at the same request, including the correct terminal decision (App. G). A deployed system legitimately holds this information when a request recurs, and success-only memory discards it. Three observations argue against verbatim copying. First, the same memory yields no gain under NonThink deployment ($+0.67$ pp, §5.3), although a non-thinking model could copy a terminal string just as easily. Second, the agent still runs its own tool calls, and it recovers the Figure 1 task on 4 of 5 seeds, not 5 of 5. Third, only $2.4\,\%$ of SETA's correct-outcome attempts skip a required tool call, close to $1.8\,\%$ without memory. We therefore scope SETA as verified-outcome reuse for recurring requests and make no claim for unseen requests (held-out check: App. P).

Cost Decomposition. SETA pays Think-mode offline exploration ($K = 4$) once and adds embedding, retrieval, and a rendered prefix online (accounting: App. A.6). This prefix raises orchestrator-call prompt tokens by $+170.9\,\%$ (App. A), already included in §7's $T^* \approx 1.4$ break-even.

4 Experimental Setup

We evaluate the same 35B open-weight mixture-of-experts backbone on two AgentArch (Bogavelli et al., 2025) (an enterprise agent benchmark) domains: requesting time off (RTO; 60 tasks, short horizon) and customer request routing (CRR; 50 tasks, long horizon). The deployed (online) attempt is Think-mode throughout the headline dispatch and the RTO retrieval-only/retrieval+EF measurements, and each domain's offline memory pool is likewise built from Think-mode exploration ($K_{\text{offline}} = 4$; App. O, App. A).

We report three serving stacks, never pooled and compared only within-stack:

- **Original**: bf16 weights, the original harness, a closed reference judge, temperature 0.
- **bf16 (corrected)**: bf16 weights, a corrected harness, an open-weight judge, temperature 0.7.
- **FP8 (corrected)**: FP8 weights, the same corrected harness and open-weight judge, temperature 0.7; replicates the retrieval-only-vs-EF contrast under both Think and NonThink deployment (§5.3).

All three stacks share the same base weights, memory pool, judge, and embedding model within their harness; the corrected stacks' open-weight judge lets anyone re-score results, and both corrected stacks verify the per-request thinking flag against the request payload.

Baselines are a workflow-memory method (AWM), an LLM-written lesson-memory method, and $K{=}4$ majority-vote decoding without memory (§5). Final-outcome success is whether the agent's final answer matches ground truth (AgentArch's `correct_final_outcome` field); strict task success additionally requires every needed tool call, no extra state-changing tool call, and fully correct arguments; despite the name, it does not check call order (App. S). All contrasts use a paired sign test and a 10,000-resample paired block bootstrap for 95% confidence intervals, corrected for multiple comparisons where noted (App. E).

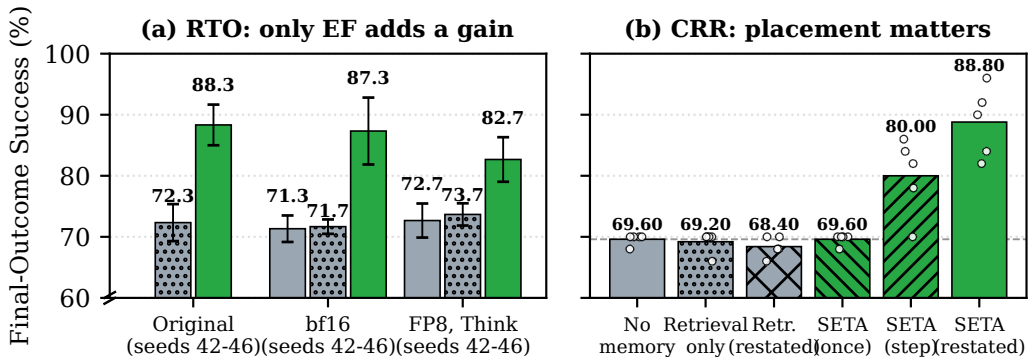


Figure 3: **Only the explicit-failure memory adds a gain, and it only helps long trajectories when shown at decision time.** Final-outcome success (%): (a) RTO by stack, mean ± s.d. over seeds {42–46} (Table 1; original's no-memory arm not measured; the three stacks are independent reproductions (harness, judge, temperature, serving stack), not a controlled single-factor ablation); (b) CRR, bf16 stack, Think, seeds {42–46}, $n = 250$ pairs/arm, points are per-seed means (Appendix W).

We use the full tool-calling environment, mature-memory configuration, and five-seed evaluation; standard $K{=}4$ controls use four attempts per task, whereas headline $K{=}1$ cells, including the headline dispatch in Table 1, use one rollout per task. Full serving, configuration, statistical, judge, and cross-benchmark details are in Appendix A.6 and Appendix A.

5 RESULTS

We test SETA's offline-amortized memory in the mature-memory, single-attempt ($K{=}1$) deployment regime described above, on RTO (60 tasks) and CRR (50 tasks), across the three stacks of §4. Figure 3 previews the two headline contrasts; the rest of this section gives each one in full.

5.1 EF MEMORY CREATES THE GAIN; RETRIEVAL ALONE DOES NOT

Table 1 reports final-outcome success by stack and mode. The headline dispatch reaches $\mathbf{85.00}\,\%$ with one rollout per task (255/300 pooled successes; per-seed counts in Appendix A); an in-family replication on a different seed set reaches $88.33 \pm 3.33\,\%$. The headline dispatch is the paper's sole canonical number.

Table 1: **Main result: final-outcome success (%) by stack and mode (RTO, 60 tasks, $K{=}1$).** Mean ± sample s.d. over seeds; the headline dispatch (bold) is the paper's sole canonical number, $88.33\,\%$ is a secondary measurement on a different seed set (seeds and Think-mode confirmation in §4, App. O). The three stacks are independent reproductions (harness, judge, temperature, serving stack), not a controlled single-factor ablation. AWM is a workflow-memory baseline (App. S); "retrieval only" includes raw-FS without EF (App. E); "—" is not measured (App. V).

Stack (seeds)	Mode	No memory	Retrieval only	Retrieval + EF (SETA)	AWM
Original ({42–46})	Think	—	72.33 ± 3.03	88.33 ± 3.33	—
Original, headline dispatch ({50, 60–63})	Think	—	—	$\mathbf{85.00 \pm 5.27}$	—
bf16 ({42–46})	Think	71.33 ± 2.17	71.67 ± 1.18	87.33 ± 5.48	—
FP8 ({42–46})	Think	72.67 ± 2.79	73.67 ± 1.82	82.67 ± 3.65	71.00 ± 3.46
	NonThink	63.67 ± 1.39	—	64.33 ± 4.80	64.33 ± 3.03

Retrieval+EF (SETA) exceeds retrieval-only by $+16.00$ pp on the Original stack (95% CI $[+11.33, +20.67]$; full bootstrap and the matched success-only-RAG control in Appendix E), and the same contrast reproduces at $+15.67$ pp on the corrected bf16 stack and $+9.00$ pp on the corrected FP8 stack; retrieval alone over no memory is not significant on either corrected stack ($+0.33$/$+1.00$ pp; §5.3). The gain also needs task-appropriate content: swapping in another task's failure diagnostic

reaches 75.67 % and a generic warning reaches 74.33 %, both indistinguishable from retrieval-only (73.67 %) and 7.00–8.33 pp below SETA (82.67 %; App. Y).

Robustness. Cross-judge agreement is high ($\kappa \in [0.986, 1.000]$, Appendix A), robust to substitution between the two evaluated judges; this is label stability, not validated correctness, since we have no human or gold adjudication. The τ-bench cross-benchmark probe supports boundary characterization rather than a positive transfer claim (Appendix I).

5.2 SETA BEATS OTHER METHODS, AT A BOUNDED COST

In Think mode, SETA exceeds AWM (a workflow-memory baseline) by $+11.67$ pp (95% CI $[+7.33, +16.33]$, $p < 0.001$), while AWM itself is statistically indistinguishable from no memory. An LLM-written lesson-memory baseline in the same slot is likewise indistinguishable from retrieval-only and from no memory, 8.33 pp below SETA ($p < 0.0001$; App. S). Four-attempt majority voting without memory matches the single-attempt no-memory baseline exactly (0.00 pp, 95% CI $[-2.33, +2.33]$), while a single SETA attempt exceeds it by $+10.00$ pp (95% CI $[+6.00, +14.00]$, $p < 0.001$; App. S): one SETA attempt beats what four attempts without memory cannot. SETA pays Think-mode exploration once and amortizes it over repeat queries, breaking even against $K{=}4$ voting at $T^* \approx 1.4$ repeat queries (§3.4), counted in backbone calls and approximate prompt tokens, not monetary cost, completion tokens, or wall-clock latency.

5.3 THE GAIN IS THINK-MODE ONLY

Under NonThink deployment, SETA reaches 64.33 ± 4.80 % against a no-memory baseline of 63.67 ± 1.39 % ($\Delta = +0.67$ pp, 95% CI $[-3.00, +4.33]$, not significant); AWM is likewise indistinguishable from both (Table 1; App. S). This Think-mode gain has a cost: the same EF-vs-retrieval-only contrast that helps final-outcome success under Think mode costs -13.33 pp of strict task success versus no memory (95% CI $[-19.33, -7.67]$), a cost shared with AWM, not unique to SETA (App. S).

5.4 MEMORY PLACEMENT MATTERS IN LONG TRAJECTORIES

On CRR (Figure 3(b)), a single SETA injection is indistinguishable from no memory ($p = 1.00$; 248 of 250 pairs tie). Restating the same memory at every orchestrator call (SETA-R) recovers $+19.20$ pp over no memory (95% CI $[+13.60, +25.20]$, $p < 0.001$); re-retrieving at every call (SETA-S) recovers $+10.40$ pp (95% CI $[+5.20, +16.00]$, $p < 0.001$). Strict task success does not improve under either placement, staying near 10 % throughout (Table 25, Appendix W). A restated retrieval-only control, repeating the same content every call without the EF diagnosis, reaches only 68.40 %. This is statistically tied with single-injection retrieval-only (-0.80 pp, 95% CI $[-2.40, +0.80]$) and 20.40 pp below restated EF (95% CI $[+14.40, +26.40]$, $p < 0.001$). The restatement-by-content interaction is $+20.0$ pp (95% CI $[+14.0, +26.0]$, $p < 0.001$). Holm-Bonferroni correction across these three comparisons keeps the EF-restated gains (P1, P2) significant; only the retrieval-only contrast (P3) is not (Table 26, Appendix W). The restated-memory gain also reproduces on a second serving stack (FP8, a different GPU type, five seeds): $+12.00$ pp (95% CI $[+8.00, +16.40]$, $p < 0.001$). Strict task success rises there too, $+6.80$ pp (95% CI $[+3.60, +10.40]$, $p < 0.001$): CRR's compliance effect is therefore stack-dependent, with no strict-success gain on the bf16 stack but a significant gain on this independent FP8 replication (Appendix W). CRR calls the orchestrator far more often than RTO does, and a diagnostic placed once is visible only at the first call; task identity, not seed or memory, otherwise sets the outcome (Appendix W). On RTO's shorter trajectories, per-step retrieval gives no significant gain over single-injection SETA ($+1.67$ pp, 95% CI $[-3.33, +6.67]$) and is not worth its added cost there (Appendix W.2). Restated memory on RTO likewise neither helps nor hurts significantly relative to single-injection SETA (86.00 % vs. 82.67 %, not significant): a single injection already reaches the decision on short trajectories, and restating does not significantly change final-outcome success on RTO (Appendix W.2).

Statistical Significance with Multiple-Comparison Correction. The full pairwise grid ($m = 56$ comparisons, both use cases), its 16 Bonferroni survivors, and mode-verification caveats are in Appendix E.

6 INTERPRETATION

Why it works. A false success looks like a success to the agent's own detector, so retrieval alone carries no warning; only an explicit, audited correction does. Think mode is what lets the agent use that correction: the same memory gives no gain once thinking is switched off (§5.3), so the diagnosis has to be reasoned over, not just copied.

When it doesn't. A single injection dilutes over a long trajectory: CRR calls the orchestrator many more times than RTO does, and a diagnostic shown only at the first call is invisible at the rest (§5.4). Whether the audited diagnosis's value transfers across benchmarks is still open, given the τ-bench substrate-density reversal (App. I).

L1: Classifier-to-Retrieval Dissociation. Classifier agreement is not enough when the label gates retrieval: an EF/TS boundary error moves a trajectory to the wrong bucket, so the high agreement reported in Appendix C coexists with downstream regression. Classifier-metric improvement is not pipeline improvement: any gating classifier must be evaluated at its downstream decision point, not in isolation.

L2: Why Success-Only Memory Fails. Success-only memory shows what worked but signals nothing about what to avoid. Raw FS is worse still: its own detector reports success, so the wrong path looks worth imitating. SETA preserves the same trajectory but relabels the outcome, without inventing one; true successes stay untouched.

7 LIMITATIONS AND CONCLUSION

Limitations. We have four scope limitations:

- **Two workflows, two roles.** RTO isolates the EF content effect under mature, same-task memory (§3.4); CRR shows the long-horizon benefit needs both persistent delivery and EF content, not repetition alone (§5.4). CRR's own single-injection mechanism ablation remains low-evidence and directional only (App. K). On CRR, a single injection has no effect; restating or re-retrieving recovers a gain at higher token cost, without improving strict success on the bf16 stack (the FP8 replication does improve it; §5.4). Claims are scoped to Think-mode deployment: the original stack's Think mode was configured and is retrospectively supported by think-trace markers, while the bf16 and FP8 stacks verify it per-request via the payload (App. O); the original stack's judge is closed, the FP8 stack's open-weight. We measure cost in backbone calls and prompt tokens ($T^* \approx 1.4$, the break-even repeat-query count against $K{=}4$ voting; App. A), not completion tokens or wall-clock latency. We have not studied adversarial or poisoned memory: an attacker who inserts a false EF record into the pool is outside this paper's threat model. Memory records can also hold request text and tool outputs (e.g., leave balances or customer records), so a deployment should apply the trajectory store's access controls, retention limits, and redaction; we did not evaluate privacy leakage and make no privacy claim.
- **Single model family.** All main-text results use one 35B open-weight MoE backbone. On a second backbone (App. F, $n = 5$), the EF-vs-retrieval-only contrast keeps a small positive direction ($+2.67$ pp), but SETA is statistically indistinguishable from its own no-memory baseline (-1.33 pp); cross-family generalization is not established. The v2 induction template encodes function-call syntax and decision-rule phrasing aligned to the primary backbone, which likely limits zero-shot transfer (App. F).
- **Compliance trade-off.** Final-outcome correctness and strict procedural compliance are separate objectives; SETA targets the former. Under Think mode on the FP8 stack, SETA's final-outcome gain (§5.3) costs strict task success, -13.33 pp versus no memory (95% CI $[-19.33, -7.67]$; App. S). These losses trace to extra state-changing tool calls (for example, a repeated notification or record update), not to skipped checks ($2.4\,\%$ vs. $1.8\,\%$ of correct-outcome attempts) or wrong arguments (none in any arm).
- **Seed-count power.** Bootstrap CIs use small seed counts ($n = 5$; some cells $n = 3$–4). Non-significant contrasts (e.g., FP8-stack NonThink) are underpowered, not demonstrated equivalence.

What this establishes. SETA rewrites confident errors into EF memories for one-attempt, Think-mode, recurring-request deployment under mature, same-task memory: it fits recurring workflows with an offline verifier, and we make no claim about unseen requests. The EF diagnosis, not retrieval, carries the gain across three stacks (§5.1), and exceeds AWM, a lesson-memory baseline, and four-attempt majority voting (§5.2). Restating or re-retrieving recovers a gain single injection misses on long trajectories (§5.4); this CRR gain, too, needs the EF diagnosis, not repetition alone (§5.4). The NonThink held-out control (App. P), where memory has no measurable final-outcome effect (§5.3), reports a -2.67 pp strict-success drop and does not bear on the Think-mode headline.

AI USE STATEMENT

We used large language models in several roles in this work. First, an open-weight language model is the tool-agent backbone under evaluation: it produced the tool calls, trajectories, and answers described in the methodology and experiments. The offline failure-memory transformation itself is *not* model-generated: it is a deterministic rewrite of already-collected trajectories and involves no language-model call. Second, an external closed reference language model served as the primary outcome judge for the TS/EF/FS/FF labels, and a second closed reference judge was used for the reported cross-judge check. Third, LLM-based coding and manuscript-editing assistance was used to help implement, edit, clarify, and organize this work. The authors designed and executed the experiments, verified the numerical analyses and results, interpreted the evidence, and approved the final claims and text. No language model is being presented as an author; the authors retain responsibility for the methodology, implementation, interpretation, citations, and final manuscript.

REPRODUCIBILITY STATEMENT

The corrected anonymized reproduction archive (`seta_supplement.zip`) accompanies this submission and contains the released implementation, configuration files, per-seed result files, judge prompts, cross-judge audit, and scripts used to regenerate the reported summaries and figures. The package also includes the trajectory-level and supplementary analyses needed to inspect the failure-memory construction, ablations, bootstrap intervals, and cross-benchmark probe. Appendix A describes the backbone, serving setup, task configuration, statistical procedure, and the released-artifact layout; the experiment configurations are under `configs/experiments/` in the released kit. The reported seed sets are $\{42–46\}$ for the standard controls and in-family replication, and $\{50, 60, 61, 62, 63\}$ for the headline dispatch. K=1 rows use one rollout per task; K=4 controls use four attempts with majority aggregation, while the headline bootstrap uses 10,000 task-level resamples of per-task deltas. The two K=1 headline figures are single-rollout-per-task measurements: the dispatch headline is supported by 255/300 pooled task successes, and the in-family replication by 265/300. The supplement includes per-seed results for both headline configurations (`supplement_headline_provenance/PAIRED_BOOTSTRAP_K1.json` and `PAIRED_BOOTSTRAP_K1_CELL3.json`). The public supplement contains per-seed outputs and analysis scripts, but a subset of per-seed raw materials is planned for the camera-ready release and is not claimed to be complete here. Reproduction of the exact reported labels depends on access to the external closed reference judge and its then-current model/API behavior; the original provider and exact version are anonymized during review, so byte-identical re-scoring is not claimed for the submission version. Results can also be sensitive to serving configuration, including the vLLM version, tensor parallelism, decoding settings, tool parser, and available hardware. The package records these configuration requirements and provides the prompts and available per-seed outputs so that reviewers can audit the analysis and rerun the pipeline with a comparable judge and serving stack.

REFERENCES

T. Bogavelli, R. Sharma, and H. Subramani. AgentArch: A Comprehensive Benchmark to Evaluate Agent Architectures in Enterprise. *arXiv preprint arXiv:2509.10769*, 2025.

B. Brown et al. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

K.-T. Cheng and A. S. Krishnakumar. Automatic functional test generation using the extended finite state machine model. In *Proceedings of the 30th Design Automation Conference (DAC)*, pages 86–91. ACM, 1993.

Anonymous. Anonymized prior taxonomy work. 2026.

P. Lewis et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *NeurIPS*, 2020.

X. Liu et al. AgentBench: Evaluating LLMs as agents. *arXiv preprint arXiv:2308.03688*, 2023.

Y. Qin et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*, 2023.

R. Rafailov et al. Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS*, 2023.

N. Shinn et al. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

X. Su, Y. Zhang, H. Luo, X. Liu, and L. Huang. Mistake Notebook Learning: Batch-Clustered Failures for Training-Free Agent Adaptation. *arXiv preprint arXiv:2512.11485*, 2025.

Y. Fu, D.-K. Kim, J. Kim, S. Sohn, L. Logeswaran, K. Bae, and H. Lee. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. *arXiv preprint arXiv:2403.08978*, 2024.

W.-C. Huang et al. A survey of agent memory in the second half: Towards self-evolving and long-horizon agents. *Transactions on Machine Learning Research (TMLR)*, 2026. arXiv preprint arXiv:2602.06052.

S. Ouyang et al. ReasoningBank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

M. Suzgun, M. Yuksekgonul, F. Bianchi, D. Jurafsky, and J. Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

X. Tang et al. Agent KB: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*, 2025.

W. Xu, Z. Liang, K. Mei, H. Gao, J. Tan, and Y. Zhang. A-MEM: Agentic memory for LLM agents. In *NeurIPS*, 2025. arXiv preprint arXiv:2502.12110.

A. Zhao et al. ExpeL: LLM agents are experiential learners. *arXiv preprint arXiv:2308.10144*, 2023.

C. Snell et al. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

X. Wang et al. Self-consistency improves chain of thought reasoning in language models. In *ICLR*, 2023. arXiv preprint arXiv:2203.11171.

G. Wang et al. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

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

S. Yao et al. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

H. Zhou, Y. Chen, S. Guo, X. Yan, K. H. Lee, Z. Wang, K. Y. Lee, G. Zhang, K. Shao, L. Yang, and J. Wang. Memento: Fine-tuning LLM Agents without Fine-tuning LLMs. *arXiv preprint arXiv:2508.16153*, 2025.

A Reproducibility Details

This appendix consolidates the configuration and reproducibility details that the main paper refers to. In appendix tables, Success (%) denotes final-outcome success; TS/FS/EF/FF denote the four-way trajectory classification used elsewhere in this paper.

A.1 Backbone and Serving

Backbone. A 35B-parameter open-weight mixture-of-experts agent backbone with $\sim$ 3B active parameters per token; the model exposes an `enable_thinking` toggle that switches between Think mode (deeper reasoning, $\sim 2.5\times$ more output tokens per attempt) and NonThink mode. We use the same backbone in both modes throughout; we never substitute a larger Think model for a smaller NonThink one.

Inference serving. Research evaluations run on a 2-GPU server (high-end accelerator class) with tensor parallelism 2 and a batch-served chat-completion API. The original-stack evaluations use temperature 0; the corrected bf16 and FP8 replications use temperature 0.7, as specified in §4 and Appendix S. Per-attempt wall-clock latencies are measured end-to-end, including the environment loop and tool calls.

Edge deployment estimate. Single-box edge deployment is feasible on a unified-memory edge accelerator at ~ 67 GB working footprint (FP8-quantized backbone weights ~ 35 GB + retrieval index ~ 2 GB + KV cache headroom). We report this as a feasibility estimate from working-memory accounting; we do not report measured throughput on the edge platform in this paper.

A.2 Closed Reference Judge Configuration

Primary judge. A closed reference language-model judge from a major commercial API serves as the primary outcome classifier c_ϕ throughout the paper. Configuration: temperature 0, single-shot prompt with structured-output decoding. The judge prompt asks for a four-way classification given the full agent trajectory and the canonical ground-truth outcome; Appendix G.3 gives the verbatim labels and the note on the prompt's own token for a false success (FS).

Cross-judge. A second closed reference judge from a different model family is used in Section 5.1 only, for inter-judge agreement measurement. Same temperature, same prompt, different model family.

Judge call statistics. Across all evaluation runs spanning every configuration and both use cases, the two judges agree closely: per-record outcome-label agreement has per-run Cohen's κ in $[0.986, 1.000]$ across the 112 runs; a pooled κ was not stored, so we do not report one. The full per-run aggregate backing this figure, 24,640 records across 112 runs, is released as `supplement_cross_judge/cross_judge_table3_aggregate.json`; a smaller worked example with per-record detail, 1,264 records across 6 runs (`cross_judge_audit.json`), is also included and is not itself the full 112-run set. The verbatim judge prompts are released as `judge_prompts/`.

Retrieval template must be set explicitly. The reported mechanism requires the enriched induction template. The generic runner's retrieval-template argument is unset by default and falls back to a skeleton-only formatter, which renders a different prompt and does *not* reproduce the reported configuration. Every reported run passes the enriched template explicitly; reproductions must do the same. The available run metadata evidence records this formatter under the field `formatter`: reported runs used `--retrieval-template ef_induction_enriched`; the inspected per-run `run_metadata.json` files do not expose a separate retrieval-template key.

Anonymous-period note. To preserve double-blind review, we anonymize the closed-judge provider names and exact model-version strings in this submission. We separate *review-time auditability* from *exact byte-level re-reproduction*: the released supplementary material includes the exact judge prompts (`judge_prompts/`), decoding parameters (temperature 0, structured-output schema), the full per-run agreement aggregate (`supplement_cross_judge/cross_judge_table3_aggregate.json`, 112 runs covering 24,640 records, with per-run κ and anonymized per-run labels), and a smaller per-record worked example (`cross_judge_audit.json`, 6 runs covering 1,264 records); raw agent trajectories and per-record raw label files beyond that worked example are not included in this anonymous review archive. This is sufficient for reviewers to (i) audit the reported per-run agreement statistics directly, and (ii) recompute the pipeline under any comparable closed reference judge (the cross-judge $\kappa \in [0.986, 1.000]$ result in Section 5.1 documents that conclusions are not coupled to a single provider's labelling, though the absolute numbers under a different judge may shift slightly). The exact provider, model name, and version string sufficient to re-run the original judge calls byte-identically are not included in the anonymous review archive; we therefore make no byte-identical re-reproduction claim during review.

A.3 Benchmark and Tasks

We evaluate two use cases of the released AgentArch benchmark Bogavelli et al. (2025); the task definitions and counts come from the repository release/configuration pinned in our code: *requesting time off* (RTO; 60 tasks; HR domain) and *customer request routing* (CRR; 50 tasks; multi-domain knowledge-base routing). The tool-call horizons we report (≤ 5 for RTO, longer multi-step routing for CRR) are observations from our pinned runs, not claims attributed to the benchmark paper. We run the benchmark's full environment loop with the function-calling agent type and its complete (non-summarized) memory mode.

A.4 Statistical Procedure

Standard K=4 controls record 4 attempts per task (pass_k $= 4$) under the standard 5 seeds $\{42\text{–}46\}$ (n=3–5 depending on configuration; per-configuration n stated in the result tables). The two reported K=1 M-MEMO-K1 headline configurations are exceptions: each records exactly one rollout per task. The headline dispatch uses non-overlapping seeds $\{50, 60, 61, 62, 63\}$, and the in-family Cell 3 replication uses $\{42\text{–}46\}$ as disclosed in Table 1 and Appendix K. The headline dispatch reports $85.00 \pm 5.27\,\%$ from 255/300 pooled task successes, and the in-family replication reports $88.33 \pm 3.33\,\%$ from 265/300 pooled task successes. Per-seed pooled counts: the headline dispatch (seeds $\{50, 60, 61, 62, 63\}$) is $\{50, 52, 54, 46, 53\}/60$, with seed 62 the minimum at $46/60 = 76.67\,\%$; the in-family replication (seeds $\{42\text{–}46\}$) is $\{52, 54, 52, 56, 51\}/60$ (recovered directly from the retained per-record CSVs, `original_stack/cell3_retrieval_ef_seed{42--46}/record_level.csv` in the released archive; these counts reproduce both the reported mean, $88.33\,\%$, and the reported sample s.d., 3.33, exactly). K=4 controls additionally use per-task continuous TS, defined as the mean over $4 \times N_{\text{seeds}}$ attempt-seed pairs. For the headline contrast, the task-paired, cross-seed bootstrap with a continuous K=4 anchor uses 10 000 resamples on per-task deltas; treatment seeds are $\{50, 60, 61, 62, 63\}$ and the K=4 anchor seeds are $\{42\text{–}46\}$. The full $m = 56$ pairwise grid (28 pairs $\times$ 2 use cases) is a separate multiplicity-corrected family, using its own paired-bootstrap procedure with Bonferroni ($\alpha/56$) and Benjamini–Hochberg (FDR $q = 0.05$) procedures.

A.5 Released Artifacts

Released as the corrected anonymized reproduction archive (`seta_supplement.zip`):

- `judge_prompts/` – the three verbatim prompt templates of Appendix G as plain-text files (`outcome_classifier.txt`, `ef_induction.txt`, `v2_retrieval_template.txt`) plus an anonymity README.
- `supplement_cross_judge/cross_judge_table3_aggregate.json` – the full per-run cross-judge agreement aggregate backing the paper's $\kappa \in [0.986, 1.000]$ claim: 112 runs / 24,640 records; judge model identifiers anonymized to `primary-judge-anon` / `cross-judge-anon` for review.
- `cross_judge_audit.json` – a smaller per-record worked example from the same agreement procedure, 6 runs / 1,264 records; not itself the full 112-run aggregate above. Raw trajectories and per-record raw label files beyond this worked example are not included in this review archive.
- `supplement_p1_1_rto_k1_ablation/` – RTO K=1 4-cell mechanism ablation (`AGGREGATE.csv` + per-seed JSONs).
- `supplement_coldstart_paired_bootstrap/` – cold-start paired-bootstrap artifacts; not used for any comparison reported in this revision.
- `supplement_block_bootstrap/` – block-bootstrap CI sensitivity for the K=1 mature-memory headline.
- `supplement_p1_2_reflexion/` – Reflexion evidence and its artifact note; the retained evidence is the five-seed RTO sanity check (App. N), not a compute-matched run. The seed-99 three-trial check is retained there only for provenance.
- `supplement_p1_3_taubench_extended/` – τ-bench airline + retail extended grid (per-seed JSON + AGGREGATE).

- `supplement_familyB_n5/` – cross-family stress-test n=5 extension.
- `supplement_hp_sensitivity/` – σ hyperparameter sweep.
- `supplement_seed_outlier_analysis/` – per-task seed outlier diagnostic for the M-MEMO-K1 dispatch.
- `supplement_awm_baseline/` – matched AWM baseline (App. S): per-seed outcomes for both metrics, the induced workflow cache, and the induction, run, and aggregation scripts.
- `supplement_bf16_replication/` – bf16-stack precision cross-check (RTO, $K{=}1$, Think, seeds $\{42$–$46\}$): per-seed outcome CSVs for the no-memory, retrieval-only, and SETA/EF-reference arms, plus the paired-bootstrap and precision difference-in-differences scripts reproducing the Table 1 precision row and the bf16-vs-FP8 DiD figure.
- `supplement_fp8_diagnostics/` – corrected-FP8-stack diagnostics (RTO, seeds $\{42$–$46\}$): per-seed outcome CSVs for the Think no-memory/retrieval-only/SETA/AWM arms (D4b/D4c/D4a/D5), the Think $K{=}4$ self-consistency-majority reference (D7, all five seeds), the three NonThink arms, and the paired-bootstrap script and aggregate JSON reproducing every reported FP8-stack delta.
- `supplement_a2_sweep/`, `supplement_b3_analysis/`, `supplement_analysis_scripts/` – supplementary sweeps and the anonymized analysis Python scripts that produce the headline statistics from the per-seed CSVs.
- `README.md` – archive overview and per-directory description.

All paths in the released artifacts are anonymized (no identifying URLs, IPs, institution names, or personal author paths). The benchmark and backbone codebases themselves (AgentArch, τ-bench, the open-weight backbone) are publicly available and are not re-bundled inside this archive.

A.6 Main-text Implementation Details

The following implementation details are moved from the main text to keep the submission within the main-text limit; no reported evidence or numerical result is removed. The full offline-to-online architecture is Figure 2 in the main text (§3).

A.6.1 Compute-role notation

Notation. K_{offline} is the number of historical Think-mode exploration trajectories collected per offline task (4 here); K_{online} is the number of live Think attempts per deployment query (1 here); and R_{eval} is the number of independent evaluation repeats (the reported standard protocol uses five seeds). Standard K=4 controls record four evaluation rollouts per task and may aggregate them by majority. The two M-MEMO-K1 headline configurations instead record one rollout per task, as stated above; their reported figures are single-rollout task-success estimates, not majority-of-four estimates. The mature-memory results compare evaluated deployment configurations with different memory visibility and attempt aggregation; they do not identify a causal effect of changing attempt count alone.

A.6.2 Offline pool construction

For each offline task we collect $K_{\text{offline}} = 4$ Think-mode trajectories from π_θ and label each by c_ϕ. Only FS cases are augmented into EF memories by an auditable deterministic rewrite:

$$g(\tau_{\text{FS}}) = \text{rewrite}(\tau_{\text{FS}}, \text{failure reason, typed arguments, provenance}) = \tau_{\text{EF}}^{(\text{synthetic})}.$$

The operation deep-copies the failed source trace, prepends its failed final answer, marks the outcome as an explicit failure, and records a diagnostic describing what went wrong and what should have been done instead. It also carries forward the typed tool arguments actually executed in the trace and records provenance from the source FS entry. The rewrite is deterministic: it does not call an LLM, re-prompt the backbone, or invent a new trajectory. Construction is performed once offline, not as a retrieval-time augmentation, so the resulting pool entry is keyed by $(\text{query}, \text{trajectory}, \text{label}, \text{failure_reason})$.

An attempted LLM re-prompt implementation produced zero valid FS→EF transitions in 12 pilots; we report that negative result in Appendix A.

We retain TS trajectories and raw FS trajectories, while adding the selected deterministic EF copies; true successes are not transformed, pre-existing EF cases are not re-induced, and FF traces are excluded from this channel. Raw FS entries remain eligible for the negative channel and are rendered under the literal "known pitfalls" header. For our two AgentArch use cases the store contains 77 negative-polarity items (the raw-FS pitfalls retrieved in §3.2) alongside 77 corrected EF items; the retriever applies no status filter beyond exclusions. The pool for each use case is built from $K_{\text{offline}} = 4$ Think-mode exploration runs of the same backbone; the on-disk trajectory store currently holds 122 RTO and 135 CRR records after this construction — fewer than the K_{offline}-based totals of 240/200 discussed below in this appendix, since not every offline exploration rollout is kept as a distinct pool entry. Pool construction is configured in Think mode throughout; reasoning text is stored inline in the final message for 59 of the 122 RTO entries ($48.4\,\%$), while the remaining RTO entries and all 135 CRR entries ($0\,\%$ with inline reasoning text) come from a serving path that returns reasoning in a separate response field rather than inline in the final message, so its absence from the stored final-message text is not by itself evidence of non-thinking generation. We assert Think mode for this pool, rather than leaving it unlabeled, because the one-directional toggle-loss defect audited in Appendix O can only convert an intended non-thinking call into Think execution and never the reverse, and pool construction was configured as Think from the outset; that defect therefore gives no path by which these Think-configured offline runs could have silently become non-thinking. The pool is indexed by sentence-embedding vectors. Pool construction is single-shot.

A.6.3 Online retrieval and inference

At inference time the agent runs π_θ^T (Think mode) with $K_{\text{online}} = 1$ (one live attempt per user query). The retriever selects $top_{TS} = 3$, $top_{EF} = 2$, and $top_{neg} = 2$ with MMR ($\lambda = 0.7$) and similarity threshold $\sigma = 0.65$. The formatter renders only $1/2/1$: one top-1 TS skeleton, up to two raw-FS trajectories as known pitfalls with diagnostics, and one EF memory, for four rendered trajectories total. The two additional TS selections and one additional EF selection are not rendered. This bucketed composition replaces the earlier shorthand "top-1 retrieval", which could incorrectly suggest that a query receives one memory.

Multi-attempt majority vote ($K \geq 2$) is not used for the live deployment path because it multiplies user-facing work; the `pass_k=4` quantity in evaluation is an aggregation rule, not a claim about four live rollouts. The system is evaluated as an online configuration comparison, not as a causal recovery of a K=4 policy at K=1 cost.

A.6.4 Retrieval gate

Retrieval is gated by a coarse mode-based router: multi-agent and orchestration traces receive retrieval injection, while single-agent simple-flow traces bypass retrieval. The retriever performs bucketed MMR selection with cross-domain fallback when same-domain similarity is below $\sigma = 0.65$; $\lambda = 0.7$ enforces diversity (both fixed before reported test runs from a small development split, not tuned on reported evaluation tasks). The deployed pool size is task-bounded ($|\mathcal{M}| = 240$ for RTO, 200 for CRR) rather than retrieval-quality bounded; the retrieval channel therefore acts as a high-coverage failure-aware exemplar lookup, and our claims are about the failure-aware editing of $\mathcal{M}$ (the FS→EF transformation), not about scaling-laws of the embedding index itself. A logit-confidence gate is documented as a follow-up operating point but is not used in any headline reported here.

Semantic and tool-call gates (not used in the reported dispatch). `semantic_gate.py` and `tool_gate.py` implement a separate, action-level SOP gate (validators including a `create_case` check). Both activate only through the offline trajectory-generation script (`run_thinkmode_pipeline.py`, flags `--semantic-gate`/`--gate`, both default off) or through an unused monkey-patch helper whose own activation function is never called anywhere in the codebase. `run_agentarch_eval.py`, the orchestrator behind every reported RTO/CRR evaluation dispatch, does not import either module, and none of its `--memory-config` choices (baseline, retrieval, gate, guard, full) reference them. Both gates were therefore off for every reported RTO/CRR evaluation run.

A.6.5 QUANTITATIVE LEAKAGE AUDIT

Quantitative leakage check. M-MEMO-K1/RTO retrieval audit (n=300 task-seeds; each query: TS-precedent + EF exemplar): top-1 same-task-id rate is 70.0 % (TS channel) / 35.0 % (EF channel) / 96.7 % either — expected for transductive $|\mathcal{M}| = 240$. Conditional TS reverses sign: TS channel $+27.8$ pp (same 96.7 % vs. cross 68.9 %); *EF channel* -21.6 *pp* (same 74.3 % vs. cross 95.9 %). The asymmetric conditional pattern is consistent with failure-conditioned retrieval but does not by itself rule out substantial same-task information reuse. Per-pair audit results are recorded in an internal leakage-check log (per-pair CSV, not among the top-level archive directories listed in Appendix A.5).

A.6.6 COST DECOMPOSITION

Offline costs (one-time/domain): pool build (K=4) $\sim$ 48 s/task; $g(\tau_{\mathrm{FS}})$ (FS only) $\sim$ 6 s $\times N_{\mathrm{FS}}$ (FS rate 8–22 %); embedding $\sim$ 50 ms $\times|\mathcal{M}|$. **Online cost** (per query): embedding-similarity $\sim$ 50 ms (full retrieval-pipeline RTT $\sim$ 1.7 s); EF prefix induces verification tool-calls at inference time (Appendix H). Default c_ϕ is a closed reference judge (Appendix A); §C examines self-classifier substitution.

Pool construction is a one-time $\sim$ 50 min cost on two 80 GB data-center GPUs; amortized per-query cost $C_{\mathrm{amortized}}(T) = C_{\mathrm{on}} + C_{\mathrm{off}}/T$ has break-even $T^* = C_{\mathrm{off}}/(3C_{\mathrm{on}})$ against K=4 majority inference. We measure C_{on} and C_{off} in backbone model calls and prompt tokens (FP8 stack, RTO, seeds {42–46}): SETA (K=1) averages 14.10 backbone calls and 31,939.6 prompt tokens per task, versus 52.16 calls and 95,529.9 prompt tokens for a K=4 Think-mode self-consistency majority-vote baseline. We use this K=4 run (D7, Appendix S) as the measurement of C_{off}: it repeats the same $K_{\mathrm{offline}} = 4$ Think-mode exploration-plus-judge-labelling procedure used to build the offline memory pool (§3.1), and the deterministic rewrite $g(\tau_{\mathrm{FS}})$ itself adds zero model calls, though it is a same-day, same-stack re-measurement rather than the literal historical run that built the released pool. Under this mapping $C_{\mathrm{off}} = C_{\mathrm{on}}(K{=}4)$, so $T^* = 1/(1-\rho)$ with $\rho = C_{\mathrm{on}}(\mathrm{SETA})/C_{\mathrm{on}}(K{=}4)$; across calls versus prompt tokens and mean versus median, $T^* \approx 1.4$ (range 1.36–1.50) — from the second time a task is queried, SETA's amortized cost is cheaper than repeating K=4 majority voting, at a $+10.00$ pp final-outcome advantage over that baseline (Appendix S reports the accompanying strict-task-success trade-off). We additionally measure SETA's per-query prompt overhead directly: at the orchestrator-call level (where retrieved memory is injected), SETA's EF-reference content adds $+170.9$ % prompt tokens relative to no memory, versus $\approx +26$ % for retrieval-only or AWM content in the same slot. Prompt-token counts are a tokenizer approximation (no exact tokenizer match for the backbone was available locally, so we substitute a same-family tokenizer); completion tokens and judge-prompt tokens are not recorded by the harness and are not measured; wall-clock latency is excluded from this comparison because concurrent task execution contaminates the recorded timestamps for most arms, while call counts and token counts are unaffected by scheduling overlap; and this measurement covers RTO only.

Self-correction of prior description. An earlier description of this system's EF construction, retrieval fan-out, and raw-FS handling was inaccurate in three respects, corrected in full in Appendix O.

A.7 FULL EXPERIMENTAL SETUP

Backbone. A 35B-parameter open-weight mixture-of-experts agent backbone with $\sim$ 3B active parameters per token and an `enable_thinking` toggle (Think mode generates $\sim 2.5\times$ more tokens than NonThink). All experiments use the same backbone in both modes; we never substitute a larger Think model for a smaller NonThink one.

Serving. Research evaluations run on a 2-GPU server (high-end accelerator class) with tensor parallelism 2. Single-box edge deployment is *feasible* on a unified-memory edge accelerator at $\sim$ 67 GB working footprint (FP8 weights $\sim$ 35 GB + retrieval index $\sim$ 2 GB + KV cache); we report this as a feasibility estimate, not a measured throughput, in Appendix A.

Benchmark. In the AgentArch repository release/configuration pinned in our code, the two tool-use domains have contrasting horizons: ***RTO*** (requesting time off, 60 tasks) and ***CRR*** (customer request routing, 50 tasks). The ≤ 5 RTO and ≥ 10 CRR tool-call horizons are observations from our pinned

runs, not facts attributed to the benchmark paper. We run the full env loop with the function-calling agent type and complete (non-summarized) memory.

Configurations. The earlier eight-base shorthand across two factors is not an exhaustive account of the executed matrix. The available RTO run matrix specifies nine conditions: C1 bare K=1 baseline; C2 retrieval-only K=1; C3 SETA full K=1; C4 E3 K=4 Think with mature memory; C5 K=4 reference without memory; C6 E1 EF-on/raw-FS-off; C7 raw-FS-only; C8 E2 matched diagnostic memory; and C9 a retained memory-agent diagnostic condition (not a compute-matched competitive baseline). The two-factor shorthand lists ten possible cells (`none / memo / heldout / memo+ef / heldout+ef` × `Think / NonThink`), but the available manifests do not identify a defensible pair of excluded cells and reasons; we therefore do not assert an eight-cell enumeration here. In the shorthand, `memo` denotes the *transductive mature-memory* regime (same-task trajectories present in $\mathcal{M}$) and `heldout` denotes the *inductive strict cold-start* regime (same-task trajectories excluded from $\mathcal{M}$). The mode factor is `Think / NonThink`. One cell receives special attention as the headline production configuration:

- **M-MEMO-K1**: Think + retrieval + EF + memo + K=1 — the production-motivated mature-memory configuration.

Statistical protocol. Standard K=4 controls use 4 attempts per task (pass_k $= 4$) under the standard 5 seeds $\{42\text{–}46\}$. The two reported K=1 M-MEMO-K1 headline configurations are the exception: each uses exactly one rollout per task. The headline dispatch $\{50, 60\text{–}63\}$ reports $85.00 \pm 5.27\,\%$ from 255/300 pooled task successes; the in-family $\{42\text{–}46\}$ Cell 3 replication reports $88.33 \pm 3.33\,\%$ from 265/300 pooled task successes. The in-family Cell 3 measurement is reported as a seed-aligned replication used for paired mechanism ablations (Table 1). The per-seed task-success rate is the mean$\pm$ standard deviation across seeds. The K=4 controls additionally use per-task continuous TS, defined as the mean over the $4 \times N_{\text{seeds}}$ attempt-seed pairs, for the headline's task-paired, cross-seed bootstrap. All confidence intervals and p-values come from the applicable per-task bootstrap (10,000 resamples on per-task deltas), not from the per-seed standard deviation. The full pairwise grid is $m = 28$ pairs $\times$ 2 use cases $= 56$ comparisons; we report uncorrected, Bonferroni, and Benjamini–Hochberg results explicitly.

Judges. A *closed reference judge* serves as the primary c_ϕ; a second *closed cross-judge* from a different model family corroborates inter-judge agreement. Exact model versions, temperature, and prompt are listed in Appendix A; both judge calls are deterministic (temperature 0) and can be reproduced from the released judge prompts.

Cross-bench probe. We additionally apply SETA's FS detector + transformation g to a held-out 50-record subset of an unrelated tool-use benchmark (airline domain) to test whether the FS→EF conversion mechanism transfers outside AgentArch.

Stack comparison. Results in this paper draw on three independent serving stacks: the original stack, the bf16 stack, and the FP8 stack. The original stack described above serves the backbone via a batch-served chat-completion API at temperature 0 on a 2-GPU server (tensor parallelism 2) and uses the closed reference judge of Appendix A.2 as the outcome classifier c_ϕ; it backs the headline dispatch (seeds $\{50, 60\text{–}63\}$) and the RTO Cell 2/Cell 3 measurements (seeds $\{42\text{–}46\}$), and its per-request thinking-mode toggle was not verified at request time (Appendix O). The bf16 stack (Appendix S) re-runs the corrected harness on the same original bf16-precision backbone weights via vLLM $0.23.0$ on a 2-GPU server (tensor parallelism 2), substitutes an open-weight judge for the closed reference judge, and samples at temperature 0.7 rather than temperature 0; it evaluates Think-mode deployment on the $\{42\text{–}46\}$ seed range. The FP8 stack (Appendix S) serves the same backbone family via vLLM $0.27.1$ with FP8-quantized weights on a single-GPU server (tensor parallelism 1), likewise substitutes the open-weight judge and samples at temperature 0.7; it evaluates both Think-mode and NonThink-mode deployment on the same $\{42\text{–}46\}$ seed range. The bf16 and FP8 stacks both verify the requested thinking-mode flag via the forwarded request payload on every call (Appendix O). All three stacks share the same original backbone weights, the same offline memory pool, and the same retrieval embedding model; the bf16 and FP8 stacks additionally share the same open-weight judge. The three stacks are independent reproductions on different serving infrastructure and precision, not a controlled single-factor ablation. The bf16 and FP8 stacks' vLLM servers both run with prefix caching and chunked prefill enabled (`--enable-prefix-caching --enable-chunked-prefill`).

B TEMPLATE ABLATION: g TRANSFORMATION VARIANTS

This appendix documents the failed LLM re-prompt pilot and the deterministic g transformation introduced in Section 3. The reported EF pool is not produced by a template call.

Variants compared.

- **LLM re-prompt pilot (negative result).** A 7-line Jinja2 prompt re-prompted the backbone with the failed final answer and asked for a one-sentence `failure_reason`; across 12 pilots it produced zero valid FS→EF transitions, so it is not used to build the reported pool.
- **Deterministic rewrite (default in main experiments).** A deep copy of the FS trace with only the terminal finish message and outcome metadata rewritten, with provenance recorded.
- **Not used.** No LLM-generated EF trajectory enters the reported pool.

Construction audit. The deterministic rewrite preserves the source trajectory and records its FS provenance. It requires no LLM call and is therefore auditable and reproducible at construction time.

Negative result. The LLM re-prompt route is retained only as a documented failed pilot: it yielded zero valid transitions in 12 trials and is not evidence for the deterministic pool.

Token cost. The deterministic rewrite adds no LLM token cost at construction time. Any storage overhead is the rewritten terminal metadata and provenance record.

Token Usage and Tool Call Count. For completeness, NonThink mode generates ~ 1.2K output tokens per online query at K=1 and Think mode generates $\sim$ 3K; RTO has at most 5 tool calls per task and CRR at least 10. These are setup descriptors, not latency measurements.

C SELF-CLASSIFIER: FULL ABLATION

This appendix gives the full self-classifier viability sweep behind the summary below.

Summary. The backbone agrees with the closed judge at classifier level ($\kappa = 0.727$, $n = 150$, $100\,\%$ FS recall), but pipeline regresses -10 to -18 pp as boundary noise changes retrieval. Routing top-quartile-entropy cases to the judge gives RTO $78.33 \pm 3.65\,\%$/CRR $70.00 \pm 2.19\,\%$ at $\sim 25\,\%$ judge cost (full sweep below).

Setup. A 30-trajectory subset (TS:FS:EF stratified 10:10:10) is labeled by both the closed reference judge and the backbone in self-classification mode under five seeds, yielding $n = 150$ pooled comparisons.

Classifier-level agreement. Cohen's κ for FS detection (binary collapse: FS vs. not-FS) is 0.727 ± 0.078 across seeds (pooled $n = 150$). FS *recall* is 1.000 in all five seeds: the backbone catches every reference-judge FS case. Per-class precision/recall are reported in the supplementary JSON; the conservative bias of the self-classifier is what produces the perfect FS recall (false positives on EF/TS rather than false negatives on FS).

Pipeline-level result (the binding constraint). Despite classifier-level viability, end-to-end SETA with the self-classifier as c_ϕ regresses by -10 to -18 pp on RTO and CRR mature-memory cells against the same pipeline with the closed reference judge. The regression is not resolved by improving classifier-level agreement: increasing κ from 0.43 under a stricter classification prompt to 0.79 with a few-shot prompt and $K{=}4$ self-consistency voting does not recover pipeline performance. The dominant failure mode is that residual EF/TS boundary noise in the self-classifier propagates into the retrieval channel: spuriously labeled EFs displace true EFs in the top-k retrieval slots.

Plan A: classifier-side mitigation. A few-shot prompt with $K{=}4$ self-consistency voting raises classifier κ from 0.43 to 0.79 but does not recover pipeline TS (regression of -15 pp on the M-MEMO setup, $n{=}1$). This is direct evidence that classifier-level metrics alone do not predict pipeline behavior.

Plan B: retrieval-side direction. A confidence-adaptive top-k retrieval modification (top-k adjusts on the predicted-confidence quantile) gives a $+2.33$ pp mean delta over the K=1 mature-memory baseline at $n = 5$ matched seeds, with paired-bootstrap $95\,\%$ CI $[-1.67, +6.33]$ pp and $p = 0.26$ (not statistically robust). The variant does *reduce* seed-level standard deviation from 5.27 pp to 3.45 pp ($-35\,\%$), so we report it as a stability-direction rather than as a verified accuracy mitigation.

Recommended hybrid recipe. For practitioners targeting reduced external-judge dependence, we recommend a hybrid recipe: route the top-quartile self-classifier-entropy trajectories ($\sim 25\,\%$) to the closed reference judge and accept self-classifier labels for the remainder. Measured pipeline performance is RTO M-MEMO-K1 $= 78.33 \pm 3.65\,\%$ (per-seed $\{76.67, 83.33, 73.33, 81.67, 76.67\}$) and CRR M-MEMO-K1 $= 70.00 \pm 2.19\,\%$ (per-seed $\{72, 70, 66, 72, 70\}$) at $n{=}5$ seeds each, with $\sim 75\,\%$ reduction in external-judge calls relative to the closed-reference baseline. The hybrid pipeline recovers most of the full-self-classifier -10 to -18 pp regression on RTO; on CRR the hybrid sits closer to the baseline regime, consistent with CRR's longer-horizon difficulty.

Methodological lesson. The relationship between classifier-level κ and pipeline-level TS is mediated by the retrieval channel's sensitivity to label noise on a small subset of borderline trajectories. We believe this lesson generalizes: when an LLM-judged label feeds a downstream selection mechanism (retrieval, voting, ranking), classifier κ alone is not a sufficient deployment proxy and pipeline-level evaluation is required.

Routing-fraction sweep. To trace the full cost-accuracy frontier behind the 25 % recipe, we extend the routing fraction over $\{0, 10, 40, 55\}\,\%$ (5 seeds each, RTO mature-memory, $n{=}20$ new runs) on top of the existing 25 % measurement. Table 2 reports the sweep. The four new operating points form an observed monotonic-then-drop pattern in this single-dispatch sweep: TS rises from $75.00 \pm 1.05\,\%$ at 0 % (self-only) through $81.67 \pm 2.11\,\%$ at 10 % to a peak of $90.67 \pm 3.09\,\%$ at 40 % routing, then drops slightly to $86.33 \pm 1.94\,\%$ at 55 %. The empirical judge-call fraction tracks the target routing budget within sampling noise (0.008 / 0.107 / 0.402 / 0.557 vs. targets 0 / 10 / 40 / 55 %), confirming the sweep mechanism. The 25 % prior measurement (78.33 ± 3.65) was obtained in a separate dispatch on a different pool build and sits below the new 10 % measurement; we include it as a consistency-check row rather than an interpolated data point on the new frontier.

Table 2: Hybrid self-classifier routing-fraction sweep (RTO mature-memory, 60 tasks, 5 seeds, K=1). Rows 1, 3, 5, 7 are this sweep ($n{=}20$ new runs, single dispatch, identical pool builds); the 25 % row is the prior measurement on a different pool build, included as a consistency check; the 100 % row is the closed-reference external anchor (separate AgentArch full-loop pool, not the hybrid pipeline; cross-pool descriptive comparison).

Routing → judge	τ_h pctile	n_{seeds}	Success (%) ± std	Judge-cost frac	Source
0 % (self-only)	100	5	75.00 ± 1.05	0.008	this sweep
10 %	90	5	81.67 ± 2.11	0.107	this sweep
25 %	75	5	78.33 ± 3.65	0.250	prior result (different pool)
40 % ***(peak)***	**60**	5	$\mathbf{90.67 \pm 3.09}$	0.402	this sweep
55 %	45	5	86.33 ± 1.94	0.557	this sweep
100 % closed-reference	—	5	85.00 ± 5.27	1.000	external anchor (different-pool)

Reading. The four-point sweep traces the same monotonic-then-drop pattern in this exploratory single-dispatch sweep on RTO mature-memory: TS rises from $75.00 \pm 1.05\,\%$ at 0 % routing (self-classifier only) through $81.67 \pm 2.11\,\%$ at 10 % routing, peaks at $\mathbf{90.67 \pm 3.09}\,\%$ at 40 % routing, then falls back to $86.33 \pm 1.94\,\%$ at 55 %. The 40 % routing point yields the best mean TS in the sweep at less than half the judge cost of the 100 % closed-reference anchor. We do not claim 40 %-vs-100 % as a strict head-to-head outcome — the 100 % closed-reference comes from a separate pool build (AgentArch full-loop), so the 90.67 vs. 85.00 contrast is an upper-bound descriptive comparison

rather than a like-for-like inference. The empirical sweet spot for deployments targeting reduced external-judge dependence on RTO mature-memory therefore appears to be in the 25–40 % routing range; CRR was not re-swept under the same protocol and is straightforward follow-up.

Inverted-U: a plausible mechanism (n=5 single-dispatch sweep, not separately verified). The peak-then-drop pattern between 40 % and 55 % routing is *consistent with* (not established by) the following hypothesis: at moderate routing fractions, the highest-entropy self-classifier trajectories are correctly escalated to the closed reference judge, yielding clean EF supervision on exactly the borderline subset that needs it; at higher routing fractions, additional self-classifier labels with progressively lower entropy (i.e., the next-most-confident trajectories) are also pulled into the routed pool, and a fraction of these are inherently noisier in their EF/TS boundary, which would offset part of the gain from reference-judge supervision on the highest-entropy subset. We frame this as a hypothesis rather than a verified mechanism: the present sweep is single-dispatch with n=5 seeds per routing fraction; multi-dispatch replication, a finer routing-fraction grid (e.g., 30, 35, 45, 50 %), and per-trajectory entropy/label-noise audits would be needed to pin down the inflection.

D Identity Metrics: Full Decomposition (and Visual Companions to Section 5)

This appendix gives the full identity-metric sweep behind the summary below, and also collects a verified result-summary figure and table that complement Section 5: the cross-stack main-result bar chart (Figure 4) and the same-mode Failure-Recurrence-Rate/Update-Yield comparison (Table 3).

Summary. For RTO ($n = 5$), without weight updates, SETA resolves 13 of the 18 tasks failing under the original stack's retrieval-only baseline, with no regressions (Update Yield 13; Failure Recurrence Rate 27.78 %). The rest of this appendix repeats this on the second stack (the DPO (Rafailov et al., 2023) value there is descriptive only).

Table 3: **Failure Recurrence Rate (FRR) and Update Yield, recomputed under matched-mode baselines (RTO, K=1, n=5 seeds $\{42$–$46\}$ per arm).** FRR and Update Yield are defined in the Method paragraph below (cross-seed majority-fail, threshold $\lceil 5/2 \rceil = 3$ of 5 seeds). The manuscript's earlier FRR/Update-Yield figure paired the M-MEMO-K1 headline (confirmed Think, App. O) against a baseline labelled NonThink at the time (Cell 1, a derived per-attempt anchor from the K=4 sweep behind Table 1; App. K does not label that sweep's mode); that cross-mode-labelled pairing is withdrawn. All rows below instead compare a memory arm (SETA) against a baseline arm run in the *same* mode; a third stack (bf16, Appendix S) reproduces the retrieval-only-baseline contrast at the paper's original weight precision on the corrected harness, and the FP8 stack is reported against two different baselines to mirror both the original stack's retrieval-only contrast and a stricter no-memory contrast.

Comparison (mode)	$\lvert F_{\text{base}} \rvert$	$\lvert F_{\text{SETA}} \rvert$	FRR (%)	Update Yield	Regressions
Original stack: Cell 3 vs. Cell 2, retrieval-only baseline (Think)	18	5	27.78	13	0
bf16 stack: P2 vs. P3, retrieval-only baseline (Think)	17	6	35.29	11	0
FP8 stack: D4a vs. D4c, retrieval-only baseline (Think)	16	12	68.75	5	1
FP8 stack: D4a vs. D4b, no-memory baseline (Think)	16	12	68.75	5	1

Reading. Under a same-mode baseline, SETA's memory resolves a majority of the base arm's cross-seed failures on the original stack (13 of 18, FRR 27.78 %) and on the bf16 stack (11 of 17, FRR 35.29 %, zero regressions), but a minority on the FP8 stack (5 of 16, FRR 68.75 %, with task 49 newly regressed). The bf16 stack reproduces the original stack's retrieval-only-baseline contrast on the corrected harness at the paper's original weight precision, and its FRR falls between the original stack's and the FP8 stack's; we do not claim a single FRR number transfers across stacks or precisions. This FP8-stack reading is unchanged whether the baseline is D4c (retrieval-only, mirroring the original stack's Cell 2 contrast) or D4b (no-memory): the two baseline arms differ in mean success (73.67 % vs. 72.67 %) but happen to share an identical cross-seed majority-fail task set at this seed count, so both baselines give the same $\lvert F_{\text{base}} \rvert = 16$, the same 16 task IDs, and hence identical FRR/Update-Yield/regression figures against D4a. We report both rows rather than only

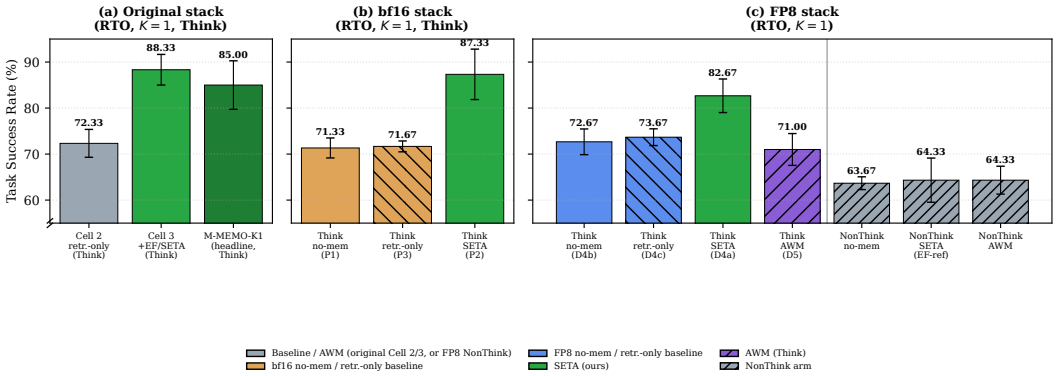


Figure 4: **Main results across three independently mode-confirmed stacks (RTO, $K{=}1$; visual companion to Table 1).** Every bar shown is confirmed single-mode: by final-message think-trace markers (App. O) for the original stack, and by request-level audit (`model_request_audit.jsonl enable_thinking_forwarded`) for the bf16 and FP8 stacks. No mixed-mode bar is shown, and the inductive strict cold-start regime is omitted from this revision (no raw per-request data survives to audit its mode, and its comparison baseline is a separate, unlabeled-mode quantity). **(a)** Original stack, Think mode throughout: Cell 2 (retrieval-only) and Cell 3 (retrieval+EF/SETA) are fresh in-family $K{=}1$ measurements on seeds $\{42\text{–}46\}$ (App. K); M-MEMO-K1 is the headline dispatch on seeds $\{50, 60\text{–}63\}$ (Table 1). **(b)** bf16 stack, Think mode, seeds $\{42\text{–}46\}$: no-memory ($71.33\,\%$), retrieval-only ($71.67\,\%$), and SETA/EF-reference ($87.33\,\%$) reproduce the original stack's Cell 2/Cell 3 contrast on the corrected harness at the paper's original weight precision; the EF-diagnostic marginal (SETA $-$ retrieval-only) is $+15.67$ pp (95% CI $[+11.33, +20.33]$), while retrieval alone contributes $+0.33$ pp (95% CI $[-2.67, +3.33]$, not significant). **(c)** FP8 stack, seeds $\{42\text{–}46\}$: Think no-memory (D4b, $72.67\,\%$), Think retrieval-only (D4c, $73.67\,\%$), Think SETA/EF-reference (D4a, $82.67\,\%$), and Think AWM (D5, $71.00\,\%$) isolate the EF-diagnostic marginal from bare retrieval exposure and from a workflow-memory baseline; D4a vs. D4b differs by $+10.00$ pp (95% CI $[+6.00, +14.00]$), and D4a vs. D5 by $+11.67$ pp (95% CI $[+7.33, +16.33]$). The three NonThink arms (no-memory, SETA EF-reference, and AWM, all $n{=}5$) are statistically indistinguishable from one another on final-outcome correctness (App. S). **Note:** the y-axis starts at $55\,\%$, not $0\,\%$, to resolve the error bars (break mark, panel a). Error bars: ± 1 sample standard deviation across seeds.

one so the FP8-stack comparison is legible against either the original stack's retrieval-only-baseline convention (D4c row) or a stricter memory-vs.-nothing convention (D4b row). All three stacks' FRR values are well above $0\,\%$ (perfect carry-over) and below $100\,\%$ (no effect). The DPO (Rafailov et al., 2023) external reference and the $K{=}4$ heldout-baseline row from the withdrawn figure are not reproduced here: both were scaled against the withdrawn $29.4\,\%$ figure and are not automatically meaningful against any value above.

Definitions (from the prior taxonomy work (Anonymous, 2026)). For a policy transition $\pi_{\text{base}} \to \pi_{\text{SETA}}$ with task-level failure sets F_{base} and F_{SETA}:

$$\text{FRR} = \frac{|F_{\text{base}} \cap F_{\text{SETA}}|}{|F_{\text{base}}|}, \qquad \text{Update Yield} = |F_{\text{base}} \setminus F_{\text{SETA}}|.$$

THR (Trajectory Hit Rate) is the fraction of online queries for which $r(q, \mathcal{M})$ returns at least one match. SRR (Safe Reuse Rate) is the conditional probability of a correct outcome given a retrieval hit.

Method. We define $F_{\cdot}$ via per-seed K-attempt majority followed by cross-seed majority-fail: a task t is in $F_{\cdot}$ if at least $\lceil n_{\text{seeds}}/2 \rceil$ seeds fail at t under per-seed K-attempt majority. This is the strictest reasonable task-level definition. All comparisons in Table 3 use $K{=}1$, where per-seed K-attempt majority reduces to that seed's single-attempt outcome (`correct_final_outcome`); with $n_{\text{seeds}}{=}5$ the majority-fail threshold is $\lceil 5/2 \rceil = 3$ seeds.

Per-task win/loss decomposition.

- **Original stack, Cell 3 vs. Cell 2, retrieval-only baseline (Think, seeds $\{42$–$46\}$).** F_{base} (Cell 2 cross-seed majority-fail, n=18): tasks $\{8, 9, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55\}$. Newly resolved by Cell 3 (n=13): $\{8, 9, 40, 41, 42, 43, 47, 49, 51, 52, 53, 54, 55\}$. Recurred ($n$=5): $\{44, 45, 46, 48, 50\}$. Regressed: none.
- **bf16 stack, P2 vs. P3, retrieval-only baseline (Think, seeds $\{42$–$46\}$).** F_{base} (P3 cross-seed majority-fail, n=17): tasks $\{8, 9, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55\}$. Newly resolved by P2 ($n$=11): $\{44, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55\}$. Recurred ($n$=6): $\{8, 9, 41, 42, 43, 45\}$. Regressed: none.
- **FP8 stack, D4a vs. D4c, retrieval-only baseline (Think, seeds $\{42$–$46\}$).** F_{base} (D4c cross-seed majority-fail, n=16): tasks $\{8, 9, 41, 42, 43, 44, 45, 46, 47, 48, 50, 51, 52, 53, 54, 55\}$. Newly resolved by D4a (n=5): $\{9, 50, 51, 54, 55\}$. Recurred (n=11): $\{8, 41, 42, 43, 44, 45, 46, 47, 48, 52, 53\}$. Regressed ($n$=1): $\{49\}$.
- **FP8 stack, D4a vs. D4b, no-memory baseline (Think, seeds $\{42$–$46\}$).** F_{base} (D4b cross-seed majority-fail, n=16): identical task set to the D4c row above. Newly resolved, recurred, and regressed task IDs are therefore identical to the D4c row above.

The recurred subsets on both stacks are a target for follow-up work (counterfactual or persona-based test-time compute on the borderline tasks). Machine-readable per-task outcomes and the computation script for this table are released alongside the supplementary archive.

Coverage of the prior taxonomy framework in this paper.

- Common operational (4/5): Task Success (Section 5), Pass@1 ($\equiv$ K=1 setting throughout), Tool Call Count and Token Usage (Appendix B). End-to-End Latency is not reported as a headline metric in this revision (App. O).
- Safety (1/4): False Success Rate (FSR; Section 3, Section J). Critical/Recoverable SOP Violation and Unsafe Replay are inherited from the prior taxonomy work and are out of this paper's method scope.
- Identity (4/4): FRR, Update Yield, THR, SRR (the summary above and this appendix).

Net coverage: $9/13$ in the present paper (4 common, 1 safety, 4 identity); of the remaining 4, End-to-End Latency is instrumented but not reported as a headline metric in this revision, and 3 are explicitly inherited from the prior taxonomy work as out-of-scope safety metrics.

E Paired-Bootstrap Multiple-Comparison Summary

Mode labels are as configured and could not be verified for these original-stack runs.

The full $m = 56$ paired-bootstrap comparison grid (28 pairs $\times$ 2 use cases, 10 000 resamples per pair) is the multiplicity-corrected family and yields the survivors below. The 16 Bonferroni-surviving pairs are listed in full, grouped by use case and ranked by p-value (all Δ in pp). The 16 surviving pairs are released in machine-readable JSON form during anonymous review at `supplement_block_bootstrap/bootstrap_ci_surviving.json`, alongside the separately reported K=1 mature-memory headline bootstrap (one pre-specified primary K=4 contrast plus three exploratory contrasts) and the cold-start paired-bootstrap result (`supplement_block_bootstrap/` and `supplement_coldstart_paired_bootstrap/`, respectively). The 40 non-surviving pairs and the full per-task delta vectors will be released at camera-ready alongside judge-identifier disclosure.

CRR (long-horizon, 14 survivors).

Comparison	Δ (pp)	p-value
NonThink baseline vs. Think baseline	-3.40	< 0.0001
NonThink baseline vs. Think + retrieval + EF (heldout)	-4.80	< 0.0001
Think baseline vs. NonThink + retrieval (memo)	$+3.50$	< 0.0001
Think baseline vs. NonThink + retrieval (heldout)	$+3.40$	< 0.0001
Think baseline vs. NonThink + retrieval + EF (memo)	$+3.40$	< 0.0001
Think baseline vs. NonThink + retrieval + EF (heldout)	$+3.30$	< 0.0001
NonThink + retrieval (heldout) vs. Think + SETA (memo)	-4.10	< 0.0001
NonThink + retrieval (heldout) vs. Think + SETA (heldout)	-4.80	< 0.0001
NonThink + retrieval + EF (memo) vs. Think + SETA (memo)	-4.10	< 0.0001
NonThink + retrieval + EF (memo) vs. Think + SETA (heldout)	-4.80	< 0.0001
NonThink + retrieval + EF (heldout) vs. Think + SETA (heldout)	-4.70	< 0.0001
NonThink + retrieval (memo) vs. Think + SETA (heldout)	-4.90	0.0002
NonThink + retrieval + EF (heldout) vs. Think + SETA (memo)	-4.00	0.0002
NonThink baseline vs. Think + SETA (memo)	-4.10	0.0004

RTO (short-horizon, 2 survivors).

Comparison	Δ (pp)	p-value
NonThink + retrieval (heldout) vs. Think + SETA (memo)	-3.33	0.0002
NonThink + retrieval (heldout) vs. Think + SETA (heldout)	-2.50	0.0002

Reading. All 16 surviving pairs are sign-consistent with the main result tables. On CRR, five of the six `Think_baseline`-vs-NonThink-* pairs survive Bonferroni; mode labels for these original-stack runs are as configured and could not be independently verified (see the note above). The four SETA-vs-baseline contrasts (rows ending in "vs. Think + SETA (memo)" or "(heldout)") survive at $p \leq 0.0004$.

Direct EF marginal (Original stack), full apparatus. The mechanism contrast is visible before the broader ablations, in Table 1's Original row: retrieval+EF (SETA) exceeds retrieval-only by $+16.00$ pp, 95% CI $[+11.33, +20.67]$, sign test 53 wins/5 losses/242 ties ($n = 300$ pairs), $p < 0.0001$. This is not retrieval in general: the success-only RAG rerun (E1: no FS, no EF) reaches $71.33 \pm 4.64\%$, statistically indistinguishable from the retrieval-only condition ($72.33 \pm 3.03\%$), so the matched contrast isolates the EF diagnosis atop raw-FS, as with the in-family replication; the same contrast reproduces on two further stacks (§5.3).

F Cross-Family Stress Test and Template Sensitivity

To partially address the single-backbone scope limitation of Section 7, we run a *zero-shot stress test*: we apply the SETA pipeline — without re-tuning the v2 enriched induction template — to one open-weight backbone from a different model family than the primary backbone. We treat this as a stress test rather than a generalization claim: the v2 template (Appendix G) was hand-authored against the primary backbone's instruction-following style, and there is no a priori guarantee that the same prompt is optimal for other families.

Setup. We (i) build a fresh offline pool from the second backbone's own $K{=}4$ Think-mode exploration trajectories on the same RTO 60 tasks (no primary-backbone pool reuse), (ii) keep every other component identical to the primary experiments (same closed reference judge, same instruction-aware sentence encoder, same v2 enriched template, same online $K{=}1$ inference), and (iii) measure three online conditions: $K{=}1$ baseline, $K{=}1$ retrieval-only (no $g(\tau_{\mathrm{FS}})$), and SETA M-MEMO-K1 (full pipeline). Three seeds (42, 43, 44). A supplementary $n{=}5$ extension on the same backbone is released in `supplement_familyB_n5/`; it preserves the qualitative conclusion of the $n{=}3$ table

below (partial preservation of the EF-vs-retrieval-only contrast, with SETA indistinguishable from the Family B no-memory baseline) and is not used as a separate headline.

The second backbone is a **30B-parameter open-weight MoE model with ~ 3B active parameters per token, from a distinct family** relative to the primary backbone. Specific identifier is held during anonymous review and will be disclosed at camera-ready.

Table 4: **Zero-shot cross-family stress test on RTO mature-memory.** All rows use $K{=}1$ deployment inference. The v2 template is *not* re-tuned for the second backbone. Pool FS rate is the fraction of 240 raw $K{=}4$ Think-mode exploration trajectories that the closed reference judge labelled FS.

Backbone	Family	Base $K{=}1$	Retr-only $K{=}1$	SETA-v2 $K{=}1$	Δ vs base	Pool FS rate
Primary 35B-A3B	A	anchor[†]	~ 72.0	$\mathbf{85.00 \pm 5.27}$	$\mathbf{+13.17}$	8–22 %
Within-family A-variant (n=5, seeds 42–46)[§]	A′	80.67 ± 3.59	78.67 ± 2.67	$\mathbf{83.33 \pm 1.83}$	$\mathbf{+2.67}$	15.79 %
Second 30B-A3B (n=3, seeds 42–44)	B	65.56 ± 1.57	64.44 ± 2.83	66.67 ± 3.60	$+1.11$	33.75 %
Second 30B-A3B (n=5 extension, seeds 42–46)[‡]	B	67.00 ± 3.80	63.00 ± 5.19	65.67 ± 3.46	-1.33	33.75 %
Second 30B-A3B (DSPy/MIPROv2-tuned, n=3)	B	65.56 ± 1.57	—	$\mathbf{69.44 \pm 2.83}$	$\mathbf{+3.89}$	33.75 %

[†] Descriptive $K{=}1$ baseline anchor (App. K Cell 1, derived per-attempt from the $K{=}4$ sweep; no $K{=}1$ raw data exists for this cell and its mode is not labeled); separately verified by a fresh $n{=}5$ re-run in App. M.

Within-family scaling — smaller-but-newer A-variant (Table 4 row §). Within the same backbone family, scaling down to a smaller (35B-A3B) but newer-version A-variant (Family A′) reduces the SETA marginal to $+2.67$ pp ($n{=}5$, seeds $\{42\text{–}46\}$). The smaller marginal is attributable to a strong baseline ($80.67\,\%$): when the bare $K{=}1$ agent is already accurate, the head-room for EF-injection-driven recovery is mechanically smaller. The within-family pool FS rate is $15.79\,\%$, well within the primary backbone's 8–22 % range, so the contraction in marginal magnitude is *not* explained by a thin EF substrate. This is consistent with the boundary axes summarized in Section 7 and detailed in Section 5.1 / Appendix I: high-baseline backbones produce fewer FS trajectories per task, and the $+$EF marginal scales with the FS substrate the offline pool can capture. The within-family direction is preserved ($+$ vs base, $+$ vs retrieval-only); the magnitude is conditional on backbone capability. Full standalone treatment in App. L.

Per-family template tuning recovers part of the magnitude. Auto-tuning the induction template for the second backbone via DSPy/MIPROv2 (offline pool kept fixed; only the g template instruction is search-target) raises SETA M-MEMO-K1 from $66.67 \pm 3.60\,\%$ (v2 enriched, hand-authored against Family A) to $69.44\,\%$ (n=3 seeds), a $+2.78$ pp gain — a partial recovery toward, but well below, the primary-backbone $85\,\%$. This is consistent with partial preservation of the EF-vs-retrieval-only contrast on Family B, while the v2 induction template requires per-family adaptation; cross-family generalization is not established; a one-shot automated tuning pipeline closes part of the gap. Per-family auto-tuning is therefore not a contribution we claim but a documented straightforward extension.

Reading the table. The result *does not* support a plug-and-play cross-family generalization claim for the current v2 instantiation. On the second backbone, SETA M-MEMO-K1 wins by $+1.11$ pp over the baseline and $+2.22$ pp over retrieval-only — the direction matches the primary backbone but the magnitude is much smaller ($+1.11$ vs. $+13.17$ pp). Notably, the second backbone produces a much higher fraction of false-success outputs in the Think-mode offline exploration pool ($33.75\,\%$ vs. 8–22 % on the primary backbone), so the EF retrieval pool is well-stocked with synthetic-EF material; the modest online gain therefore is not explained by a thin EF pool.

The retrieval-only condition (no $g(\tau_{\mathrm{FS}})$) actually *regresses* slightly on the second backbone (-1.11 pp vs. baseline), while adding the EF transformation *recovers* a small positive gain ($+2.22$ pp vs. retrieval-only). This contrast is consistent with the SETA g step contributing the small positive direction on this backbone: retrieval without EF is mildly below base, and adding EF moves the result back toward base. With $n{=}5$ seeds and SETA statistically indistinguishable from the no-memory baseline, it does not establish the g step as the active component on Family B.

n=5 extension and the SETA-vs-baseline sign flip (‡). Extending Family B from the parser-clean $n{=}3$ subset (seeds $\{42, 43, 44\}$) to the full $n{=}5$ set $\{42\text{–}46\}$ shows the SETA-vs-baseline contrast

flips sign ($+1.11$ pp $\rightarrow -1.33$ pp), but **the SETA-vs-retrieval-only architecture-transfer contrast is preserved** ($+2.22$ pp $\rightarrow +2.67$ pp). Both n=3 and n=5 SETA-vs-baseline values fall within the ± 3.5–3.8 pp seed-level standard deviation, so neither n is statistically powered to distinguish a small positive from a small negative Δ on Family B. The qualitative reading at both n is identical: *SETA $\approx$ baseline on Family B without per-family template tuning, with a small positive contrast over retrieval-only that the architecture-transfer claim rests on*. This is precisely why we treat the cross-family result as "consistent with mechanism transfer but not statistically powered as a cross-family generalization claim" rather than as a positive Δ claim. Full n=5 raw artifacts are released in `supplement_familyB_n5/`.

Why the magnitude differs. The v2 enriched template encodes (i) typed tool-call skeletons in a syntax that is closest to Family A's native function-calling format, (ii) a 2-step decision rule ("STEP A / STEP B") with explicit Markdown headers that Family A's instruction-tuning closely follows, and (iii) labeller-articulated failure-reason prose with a Family A reasoning style. Family B re-renders the skeleton, soft-follows the decision rule, and articulates failure reasons in its own thinking-channel format; these mismatches accumulate in the retrieval prefix the online K=1 agent must consume.

Methodological takeaway. We separate two claims that the current paper must *not* conflate:

1. The SETA *architecture* (offline pool + FS detection + $g(\tau_{\text{FS}})$ + retrieval injection + K=1 online) shows only partial preservation of the EF-vs-retrieval-only contrast on Family B: the small positive Δ is consistent with some signal carrying across families, but SETA is indistinguishable from the Family B no-memory baseline and n=3 seeds does not support a cross-family generalization claim.
2. The current v2 *induction template* is not yet backbone-agnostic: it transfers weakly to Family B without re-tuning. Per-family prompt adaptation (e.g. DSPy/MIPROv2 with a per-backbone validation set) is a natural follow-up rather than a contribution of this paper.

We therefore decline to make a strong cross-family generalization claim. The Section 5 main result on the primary backbone remains the headline; this appendix's role is to honestly delimit the current v2 instantiation's reach and motivate per-family template adaptation as future work.

G Agent Architecture, SETA Pipeline, and Prompts

This appendix gives a reviewer three things beyond the main text. §G.1 gives the exact multi-agent setup behind every reported RTO/CRR number: which agents exist, how many tools each one has, and the one point where retrieved memory enters a prompt. §G.2 walks through what the SETA pipeline does at each stage, with statistics counted directly from the released trajectory store, logs, and code. §G.3 onward gives the verbatim text of every prompt and template behind a reported result. Every configuration value below is read from the released harness code and usecase configuration files, not restated from memory.

G.1 Agent Architecture

Every reported RTO and CRR number comes from the same harness, `scripts/run_agentarch_eval.py`, driving the released AgentArch benchmark (Bogavelli et al., 2025) unmodified. Table 5 lists every configuration choice that shapes what an evaluated attempt actually sees.

Table 6 gives the full agent and tool inventory for both use cases, read directly from the AgentArch usecase configuration files and unchanged from the public benchmark release.

RTO's 8 tool slots and CRR's 31 tool slots (30 distinct tools; `get_relevant_knowledge_articles` is used by both `case_creation_agent` and `informational_queries_agent`) are each owned by exactly one agent in the YAML; the orchestrator itself calls no tool directly, only a sub-agent by name.

Table 5: **One fixed configuration produces every reported RTO/CRR number; only the memory arm and mode (Think/NonThink) vary across rows of the main results tables.** Values are read from the harness's argument parser, the AgentArch usecase configuration, and the launch script used for the FP8-stack SETA arm (D4a).

Setting	Value	Where fixed
Orchestration	Multi-agent: one orchestrator plus one sub-agent per domain role (3 for RTO, 9 for CRR; Table 6). Dispatch mode is `direct`, not `single_agent` or `indirect`.	`--mode direct`
Tool-calling style	Function calling. Each sub-agent is exposed to the orchestrator as one callable tool with a single string argument, `main_objective`.	`--agent-type function_calling`; `get_agents_as_tools`
Memory management	`transparent`: the full, non-summarized conversation history is passed on every turn. This is AgentArch's own setting, unrelated to the SETA memory pool.	`--memory transparent`; choices `transparent`/`compact`
AgentArch thinking-tools feature	Off. This is AgentArch's own explicit-reasoning-tool option. It is separate from the backbone's own thinking mode below, and the flag is never passed in any reported dispatch.	`--thinking-tools`, absent from the SETA headline launch command
Backbone thinking mode	Forwarded per request as `chat_template_kwargs.enable_thinking`. Think mode ($K_{\text{online}} = 1$) for every reported SETA, retrieval-only, and no-memory arm.	`utils/model.py`
Retrieval/gate/guard layer	`retrieval` only: the memory-retrieval layer is on; the confidence gate and finalization guard of Appendix A are off for every reported arm.	`--memory-config retrieval`
Orchestrator loop bound	6 iterations (RTO), 18 (CRR). Each sub-agent's own loop is separately bounded at 10 (RTO) / 16 (CRR).	`orchestrator_max_iterations`

Table 6: **RTO is a short, 3-agent pipeline; CRR is a longer, 9-agent one, and both expose every tool through exactly one owning sub-agent except one knowledge-lookup tool shared by two CRR agents.** Counted directly from `requesting_time_off.yaml` and `customer_request_routing.yaml`.

Use case	Sub-agent	Tools	Role
RTO (60 tasks; orchestrator selects among 3 agents, then finishes with "Request approved" or "Request rejected")			
	user_information_retriever_agent	1	Resolves the employee's ID from their name.
	balance_checking_agent	3	Checks leave-type validity, balance, and pending requests.
	leave_approval_agent	4	Checks conflicts, routes approval, updates the record, notifies the employee.
CRR (50 tasks; orchestrator selects among 9 agents across a 7-step SOP, then finishes with "Case created" or "No new case created")			
	request_validation_agent	5	Validates the request and extracts customer/sentiment context (step 1).
	duplicate_detection_agent	2	Flags and links duplicate records (step 2).
	case_creation_agent	8	Creates and files a case for negative-sentiment or complex-issue requests (steps 3, 6).
	informational_queries_agent	3	Answers informational intents (step 4).
	transactional_query_responder_agent	2	Handles transactional intents (step 5).
	intent_and_sentiment_extraction_agent	2	Extracts intents and sentiment from the request.
	entity_extraction_agent	4	Extracts entities and required fields from the request and its documents.
	document_verification_agent	3	Checks required documents and their status.
	email_agent	2	Drafts and sends the customer-facing email response (step 7).

Memory injection point. Retrieved memory enters a prompt at exactly one place: the orchestrator's very first turn. Figure 5 traces the path from the usecase configuration to the two prompt templates an evaluated task actually issues (§G.7, §G.8).

Both prompt templates are given verbatim in §G.7–§G.8.

G.2 SETA Pipeline: Stage-by-stage Behavior and Measured Statistics

This section walks through the SETA pipeline in the order it runs: offline pool construction, then online retrieval, then generation. Numbers already reported elsewhere are cited, not repeated. New numbers are counted directly from the released trajectory store and evaluation logs.

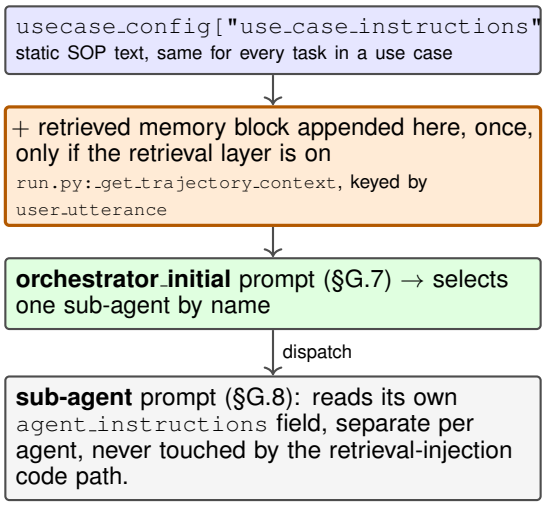


Figure 5: **Retrieved memory can only reach the orchestrator's first turn; it cannot reach a sub-agent turn, by construction.** `agent_arch/run.py` appends the retrieval text to a local `usecase_directions` variable used only by the `orchestrator_initial` template (lines 118, 131–135). Each sub-agent instead reads `self.agent_config["agent_instructions"]` via `set_agent_instructions()` (`agent_arch/agent.py:123--125`), a distinct per-agent configuration field that the retrieval code never writes to.

Offline: pool composition. The offline memory pool $\mathcal{M}$ is built once per use case from $K_{\text{offline}} = 4$ Think-mode exploration trajectories, labeled by the outcome classifier c_ϕ (§G.3), and stored with an `outcome` field the retriever reads directly. Appendix A already reports the on-disk totals (122 RTO, 135 CRR records) and the deployed pool sizes (240 RTO, 200 CRR). Table 7 adds the split by outcome bucket.

Table 7: **True successes (TS) dominate the RTO pool; explicit failures (EF) and their matched negatives dominate CRR's.** Counted directly from the `metadata["outcome"]` field of every on-disk record in the released trajectory store. EF and negative counts are equal within each use case because every EF record is the deterministic rewrite of exactly one retained negative record (§G.10).

Use case	TS	EF	Negative (raw FS)	Total
requesting_time_off	76	23	23	122
customer_request_routing	27	54	54	135

Offline: the domain filter, not embedding quality, is the safety net. The retriever pools candidates from every use case, not only the query's own domain, then applies a same-domain-first filter: if the best same-domain similarity clears $\sigma = 0.65$, cross-domain candidates are never considered (Appendix A). A separate retrieval-sensitivity check replays all 60 RTO queries against the full 257-record pool (RTO plus CRR) under 6 embedding-model and instruction variants. Domain purity, the fraction of top-1 hits landing in the query's own use case, is 1.000 for every variant: the structural filter makes domain accuracy insensitive to embedding choice in this pool. The margin between the top-1 and second-place candidate is not insensitive: it ranges from 0.53 down to 0.15 across the 6 variants, over $3\times$ narrower for one instruction-tuned embedding variant than for the production choice. Because bucket accuracy stays perfect while the margin thins for that variant, this is a latent robustness concern for a harder or more balanced pool, not an observed failure in the reported pool. Changing only the instruction wording, holding the embedding model fixed, also changes which trajectory is retrieved far more than it changes any accuracy metric: two alternative instruction wordings each retrieve a different top-1 true-success trajectory than the production wording on roughly 36–60 % of the 60 RTO queries, despite identical domain purity and hit rate.

Offline: the EF rewrite. Every EF record is produced by the deterministic rewrite rule of §G.10, which gives the rule and one concrete before/after example.

Online: retrieval and generation cost, per arm. Table 8 extends the SETA-versus-$K{=}4$ headline pair already reported in Appendix A with the intermediate arms.

Table 8: **Adding the explicit-failure (EF) diagnostic costs far more per task than plain retrieval, in both backbone calls and prompt tokens.** FP8 stack, RTO, $K{=}1$ unless noted, seeds $\{42–46\}$ ($n{=}300$ task–seed pairs per arm). Prompt tokens are a same-family tokenizer approximation, not an exact count (Appendix A). D7 sums four attempts per task.

Arm	Backbone calls/task (mean/median)	Prompt tokens/task (mean/median)	Judge calls/task
No memory	13.37 / 14.0	25,253.9 / 26,432.5	1.00
Retrieval-only (no EF)	12.94 / 15.0	23,088.4 / 27,049.0	1.00
AWM (§G.11)	13.84 / 15.0	25,819.5 / 27,779.5	1.00
SETA (EF-reference)	14.10 / 15.0	31,939.6 / 32,135.0	1.00
$K{=}4$ no-memory majority vote (4 attempts summed)	52.16 / 56.0	95,529.9 / 104,608.5	4.00

The extra cost is concentrated at the orchestrator's own call, the one place the retrieved-memory block is injected (§G.1): Appendix A reports a $+170.9\,\%$ prompt-token increase there for SETA against $\approx +26\,\%$ for retrieval-only or AWM content in the same slot. SETA also issues more orchestrator round trips than no memory: 4.127 versus 3.453 per task ($+19.5\,\%$), the measured counterpart of the qualitative claim in Appendix A that the EF prefix induces extra verification tool calls (case studies in Appendix H).

G.3 Outcome Classifier c_ϕ Prompt (per-trajectory judge)

This appendix releases the verbatim prompt templates used in our experiments: the outcome-classifier c_ϕ prompt below; the piloted but unused $g(\tau_{\mathrm{FS}})$ **ef_induction** prompt (§G.4); the **v2 enriched** retrieval-injection template (§G.5); the orchestrator and sub-agent system prompts (§G.7–§G.8); the RTO/CRR usecase directions (§G.9); the EF rewrite rule and example (§G.10); the D8 lesson-notebook prompt (§G.6); the AWM induction and injection prompts (§G.11); the evaluation-time final-outcome judge prompt (§G.12); the Reflexion reflection prompt (§G.13); the content-specificity control text (§G.14); and the $K{=}4$ majority-vote rule (§G.15). The reported EF pool uses the deterministic offline rewrite described in Appendix A.6, not the ef_induction prompt. All prompt text is released in the supplementary code under `judge_prompts/` and inside the released SETA package.

A 4-way classification prompt sent to the closed reference judge for every trajectory in the offline pool. Alongside the ground-truth and agent final responses, the judge also receives a truncated trajectory trace, self-verification flags, and the trajectory's gate-block history, and replies with a JSON object giving the label (`TS`/`EF`/`SF`/`FF`) and a one-sentence reason; see the terminology note below for how `SF` relates to this paper's false-success (FS) term.

```
You classify an agent trajectory's outcome.

Definitions:
- TS (true success): the agent's FINAL response matches ground truth.
  Mid-trajectory gate blocks followed by self-correction (e.g. agent first
  attempted an action that was blocked, then reversed course to the correct
  action, and the FINAL response matches ground truth) STILL count as TS --
  the gate exists precisely to enable this self-correction; using its
  feedback successfully is a positive signal, not a failure.
- EF (explicit failure): agent did NOT solve the task but explicitly
  acknowledged limitation, refused, or handed off -- graceful failure.
- SF (silent failure): final response does NOT match ground truth, OR
  agent CLAIMED success in finish while a state-mutating tool call was
  BLOCKED by the gate (i.e. no actual mutation occurred but agent still
  claimed it did).
- FF (false failure): agent claims failure but actually matches ground truth.

Decision priority: judge by FINAL response vs ground truth FIRST. Only
downgrade to SF if the final claim is inconsistent with what actually
happened (e.g. claimed approval but mutation was blocked).

Ground truth final response: {gt_final}
Agent final response: {agent_final}
```

```
Agent trace (truncated): {trace_snip}
Self-verify issues flagged: {issues}
Gate blocks during this trajectory: {gate_blocks_snip}

Reply ONLY with JSON: {"label":"TS|EF|SF|FF","reason":"<one sentence>"}
```

A second judge from a different closed-weight model family receives a paraphrased version of the same prompt; per-record agreement between the two is reported in Section 5.1 (per-run Cohen's $\kappa \in [0.986, 1.000]$).

Terminology note. The judge prompt uses the label SF (silent failure) for what the paper calls a false success (FS).

A second, separate judge call. c_ϕ above runs once per offline exploration trajectory, at pool-construction time. A different prompt runs once per evaluated online attempt, at evaluation time, and produces the `correct_final_outcome` field this paper reports as final-outcome success throughout. Appendix T already analyzes its token cost and cites its code location; §G.12 gives its verbatim text.

G.4 $g(\tau_{\text{FS}})$ `EF_INDUCTION` PROMPT (FS→EF SYNTHESIS)

This prompt was piloted but is not the mechanism used in the reported results. It was sent to the backbone (Think mode) once per detected false success, with the intent of eliciting an EF trajectory in which the agent acknowledges the failure and emits an explicit `finish`-tool declaration of the form "Unable to ⟨action⟩ because ⟨specific reason⟩." Across 12 pilot runs it produced *zero* valid FS→EF transitions. The EF memories used throughout this paper are instead built by the deterministic rewrite described in Section 3 and Appendix O, which requires no model call. We reproduce the prompt here as a negative result.

```
You previously attempted the task below. The attempt SILENTLY FAILED
- you produced a final answer that did NOT match ground truth, but
you did not acknowledge the failure.

Task (user utterance):
{utterance}

Your previous trace (truncated):
{trace_snip}

Your previous (false success) final answer:
{prev_final}

Ground truth final response:
{gt_final}

Reason it was a false success (judge analysis):
{judge_reason}

Now perform the EF (Explicit Failure) RE-EXECUTION:
1. Identify the precise reason your previous attempt was wrong.
2. Re-attempt the task; stop the moment you can no longer satisfy
   the request.
3. Use the `finish` tool to produce a final message of the form:
   "Unable to <action> because <specific reason from step 1>."
   Do NOT silently fabricate.

Reply with:
  - One sentence stating the failure reason
  - A `finish` call wrapped as JSON:
    {"tool_name":"finish","tool_args":{"message":
     "<your explicit-failure final message>"}}
```

G.5 v2 ENRICHED RETRIEVAL-INJECTION TEMPLATE

Inserted as a prompt prefix at online inference time, between the system prompt and the user utterance, as the retrieval text of Figure 5. Selected by `--retrieval-template ef_induction_enriched` (the released harness's other choices are `skeleton_only`, the pre-v2 legacy formatter, and `ef_induction`, the v1 predecessor of this template). Each retrieved

bucket (TS / negatives / EF) is rendered with typed tool arguments and the labeller's failure reason. The template ends with a 2-step decision rule that biases the agent toward emitting an explicit-failure declaration when the precedent does not match.

```
[Retrieved Context - Enriched]

- Successful precedent (TS) -
Original task: "<user_utterance of TS exemplar>"
Steps (with concrete arg types):
  1. <tool_name>(<arg=TYPE:"value">, ...)
  2. ...

- Known pitfalls (false successes with diagnostic) -
Previous false-success final answer: "<final message>"
Trajectory divergence: <step-by-step diff vs TS>
Why this failed (judge analysis): <classifier_reason>
=> Avoid this pattern AND avoid the underlying cause above.

- Explicit failure exemplar (EF) -
Task: <user_utterance of EF exemplar>
Steps (typed): 1. ... 2. ...
Final declaration: "Unable to <action> because <reason>"
Why explicit failure was correct: <classifier_reason>
+ Correct pattern - explicit declaration with reason, not silent
  success.

[Decision rule - 2-step diagnostic]
STEP A. Does the current task structurally match the successful
        precedent above?
  - YES -> Follow it step by step, mapping arg values from the
           user_utterance.
  - NO  -> Go to STEP B.

STEP B. Does any required information remain missing OR does any
        tool result contradict an assumption the agent would need
        (cf. the 'Why this failed' lines)?
  - YES -> DECLARE EXPLICIT FAILURE via the `finish` tool with
           message "Unable to <action> because <specific reason>".
           Do NOT fabricate.
  - NO  -> Proceed with the closest matching precedent, but log
           assumptions.

Strictly preferred: explicit failure with stated reason >
silent success > guess.
```

Authoring note. All three templates above were *hand-authored* against the primary backbone's instruction-following style. The v2 enriched retrieval-injection template, in particular, was iterated by hand to address five information-loss issues we identified in an earlier v1 template (specifically: dropped `classifier_reason` from labelling, dropped typed tool arguments, dropped trajectory divergence, and Think-mode reasoning trace; the first three are addressed in v2, the last two are revisited in the DSPy/MIPROv2 ablation in Appendix F). The main-result tables in Section 5 use the hand-authored v2 enriched template; we explicitly do not claim the prompts are auto-tuned.

Terminology note. The verbatim v2 template above retains the phrase "silent success" as a behavioural-pattern label (the agent silently fabricating a success report without verification). This is semantically equivalent to a *false success* (FS) outcome under the taxonomy used in the body and other appendices: the prompt was authored before terminology was unified, and we preserve the original wording here to keep the released prompt byte-identical to what was used in our experiments.

Per-family adaptation. Cross-family transfer (Appendix F) preserves the SETA effect direction but at smaller absolute magnitude — consistent with prompt-backbone coupling. We treat per-family DSPy/MIPROv2 tuning of the retrieval-injection template as a natural follow-up rather than a contribution of this paper.

G.6 D8 Lesson-Notebook Generation Prompt

Used to construct the LLM-written lesson-memory baseline reported in Appendix S (D8), operationalized in the style of Mistake Notebook Learning (Su et al., 2025). Sent once per audited explicit-failure pool item (23 calls for RTO) as a single NonThink call from the same backbone used elsewhere in

this paper; the online agent that later reads the resulting lesson text at retrieval time runs in Think mode, matching SETA's own evaluation condition.

```
System:
You write short lessons-learned notes for a coaching notebook used by an AI agent that handles
      workplace requests. You will be shown one past attempt at a task that was later judged
     incorrect, along with the correct outcome and the reason it was judged wrong. Write
     exactly one short, general lesson (1-2 sentences) that a future attempt at a similar task
      could read beforehand to avoid repeating the same mistake. Generalize away from this
     specific person's name, dates, and numbers -- state the underlying rule, not the specific
      case. Output only the lesson text itself, with no preamble, no bullet marker, and no
     restatement of the task.

User (template):
Task given to the agent:
"{user_utterance}"

Tools the agent called, in order:
{tool_skeleton}

The agent's final answer:
"{fs_final_message}"

Correct final outcome (ground truth):
"{ground_truth_final_response}"

Why the attempt was judged incorrect:
{classifier_reason}

Write the one-sentence (or two-sentence) lesson now.
```

`enable_thinking=false` is passed explicitly via `extra_body.chat_template_kwargs` for this call.

Example output (verbatim). For explicit-failure pool item `17bf39893dbb516f`: *"Always verify the actual result of conflict detection or balance checks before assuming a request is eligible for approval. Do not automatically route a request for approval or update its status to 'approved' without confirming that no conflicts exist and sufficient leave balance is available."*

G.7 Orchestrator System Prompt

The orchestrator's first turn uses the `orchestrator_initial` template (`agent_arch/configs/prompts.yaml`), the template used whenever `--mode direct` (every reported RTO/CRR dispatch; §G.1). {`usecase_directions`} is the memory slot: retrieved memory, when the retrieval layer is on, is appended to this field before the template is filled (Figure 5). {`memory`} below is the running conversation history of tool calls so far, a separate thing from the retrieved memory block.

```
[system]
You are the orchestrator of a group of agents. Your job is to select the next agent(s) best
     suited for accomplishing the request. You must follow the directions you are given. If
     the directions instruct you to complete steps in a specific order, you must follow that
     order.
When you select an agent, you must give it a main objective that describes what the agent
     should accomplish.

You also have access to the conversation history below which shows you what agents have
     already done to accomplish the task.

Directions:
{usecase_directions}          <-- MEMORY SLOT: retrieved memory is appended
                                      to usecase_directions before this fills in

[user]
Request:
{user_utterance}

Conversation History:
{memory}

Always select an agent. You must always pass real values as arguments. You may never use
     placeholders.
Before selecting the next agent(s), consider your request, directions, and the conversation
     history thus far to understand what to do next.
```

```
If there are no relevant agents, respond with "Unfortunately no agents are able to help with
    this request." Otherwise you MUST respond with at least one agent.
```

The line marked `MEMORY SLOT` is an annotation added for this appendix, not part of the released template file; the underlying text is otherwise verbatim from `prompts.yaml`. Sub-agents are exposed to this prompt indirectly, as the "agent(s)" the orchestrator selects, formatted as callable tools (§G.1); the orchestrator never sees a sub-agent's own instructions.

G.8 Sub-agent System Prompt

Every sub-agent turn, under function-calling mode (§G.1), uses the `agent` template from the same file. {`agent_instructions`} is *that specific sub-agent's own* field from the usecase configuration (`agent_config["agent_instructions"]`, set in `set_agent_instructions()`, `agent_arch/agent.py:123--125`), for example `balance_checking_agent`'s own eligibility rule text. This is a different configuration field from the orchestrator's `usecase_directions`, and the code path that appends retrieved memory never writes to it. No retrieved-memory text reaches a sub-agent prompt in any reported run.

```
[system]
You are an agent. Your goal is to accomplish your main objective using the directions provided
    . You must follow the directions you are given. If the directions instruct you to
    complete steps in a specific order, you must follow that order.

Directions:
{agent_instructions}

[user]
Main Objective:
{main_objective}

Conversation History:
{memory}

Always call a tool. You must always pass real values as tool inputs. You may never use
    placeholders.
Before calling a tool, remember to consider the conversation history to understand what has
    already been accomplished and what needs to be done next.
When you have finished your task, you MUST select the finish tool. Otherwise your response
    will be ignored.
```

{`main_objective`} is filled in by the orchestrator's own tool call (§G.7), not by the harness directly.

G.9 RTO and CRR Usecase Directions

The static SOP text that fills {`usecase_directions`} above, before any retrieved memory is appended. Read verbatim from each usecase's `use_case_instructions` field; both are short enough to reproduce here in full.

RTO (`requesting_time_off.yaml`).

```
Help the user with their leave request. Only proceed to requesting the leave if the employee
    is eligible for leave.
Before ending execution, you MUST respond with one of the following options: "Request rejected
    " if the request was rejected or "Request approved" if the request was approved.
```

CRR (`customer_request_routing.yaml`).

```
You are tasked with routing and handling customer service requests.

Follow the high level instructions below in order to complete your task. You must follow the
    steps in the order specified.

Step 1. Customer request validation
- Validate the customers request and extracts context from the request. If the request is not
    valid, then finish execution.

Step 2. Duplicate Record Detection
- Detect if the record is a duplicate.
- If the record is not a duplicate, then continue. Otherwise link the source record with the
    duplicate case and finish execution.
```

```
Step 3. Negative Sentiment Routing and Processing
- Consider the extracted sentiment of the customer request.
- If the sentiment is negative, then create a case and skip steps 4 and 5.
- If the sentiment is not negative, then continue to step 4.

Step 4. Process Informational Intents
- Retrieve and process any informational intents.

Step 5. Process Transactional Intents
- Retrieve and process any transactional intents.

Step 6. Process Complex Issue Intents by Creating Cases
- Retrieve any existing complex issue intents. If a complex issue intent is found and the
    record table is interaction, then process them by creating a case.

Step 7. Email Response
- Generate and send an email response so that the customer is updated on the situation with
    their request.
- This is the only way for the customer to gain information on what is happening with their
    request.

You must keep selecting tools until all of these steps are completed.
Before ending execution, you MUST respond with one of the following options: "Case created" if
     a new case was created or "No new case created" if no case was created.
```

These are unmodified AgentArch benchmark text, not authored for this paper; task-level utterances and ground truths (60 RTO, 50 CRR) are far longer and are released in full in the supplementary code rather than reproduced here.

G.10 EF Deterministic Rewrite: Rule and a Worked Example

Appendix O already states the rule in prose: the rewrite deep-copies a false-success (FS) trace and deterministically edits its terminal finish message, the mirrored finish result where present, an optional trailing orchestrator summary, and the outcome/status fields, while assigning a derived trajectory identifier and recording provenance. No model call is made. Listing 1 gives the same rule as implemented; Table 9 gives one real pool record before and after.

Listing 1: Deterministic FS→EF rewrite, as implemented (`scripts/synthesize_ef_bucket.py`). No LLM call.

```
ef = deepcopy(source_FS_record)
new_message = synthesize_ef_message(gt_tag, ef)   # rule-based text
                                                   # built from fields
                                                   # already in ef's
                                                   # own trace, e.g.
                                                   # employee, leave
                                                   # type, dates,
                                                   # balance -- no
                                                   # model call
rewrite_last_finish(ef.trace, new_message)
  # overwrites: the last finish call's tool_args.message,
  #             its mirrored tool_result.message (if present),
  #             a trailing orchestrator summary string (if present)
  # unchanged:  every earlier step in the trace, byte-for-byte

ef.trajectory_id      = sha256("ef_synth::" + source_FS_record.id)[:16]
ef.metadata.outcome   = "EF"        # was source's own label (e.g. "SF")
ef.metadata.synthesis_source = source_FS_record.id   # provenance kept
ef.metadata.synthesis_method = "rule_based_ef_conversion"
ef.metadata.original_classifier_label  = source_FS_record.metadata.label
ef.metadata.original_classifier_reason = source_FS_record.metadata.reason
```

The rewrite does not re-simulate the trajectory: the tool calls that follow `route_for_approval` in the trace, including `update_leave_record` with `status='approved'`, are copied unchanged even though the new final message declares rejection. Only the terminal declaration and the outcome metadata change; this is what "deterministic rewrite" means throughout this paper, not a re-run of the episode.

Table 9: **The rewrite changes only the final declaration; every tool call before it is untouched.** One real RTO pool record (source id `1eed3c7dad40826a`, EF id `17bf39893dbb516f` = `sha256("ef_synth::1eed3c7dad40826a")[:16]`), read directly from the released trajectory store. The employee name is synthetic benchmark mock data, the same kind already published in Appendix H.

Field	Value
Task	"i'd like to request time off from february 10th through april 2nd 2024 for holiday"
Tool trace	Unchanged by the rewrite: resolve employee ID, validate leave type, fetch balance, check pending leaves, detect conflicts, route for approval, update the leave record, notify the employee.
Before (source, classifier label SF)	Final message: *"Request approved."* Classifier reason: *"The final response was 'Request approved' but the ground truth final response is 'Request rejected,' so the outcome is a silent failure."*
After (rewrite, label EF)	Final message: *"Unable to approve vacation leave for Lisa Garcia from 2024-02-10 to 2024-04-02: request does not satisfy policy (available balance 67 days, requested 53 days; per SOP the request must be rejected). No leave record was updated. Request rejected."*

G.11 AWM Workflow Induction and Injection Prompt

The matched AWM baseline of Appendix S induces one reusable workflow per retained true-success (TS) trajectory, via one LLM call each, cached to disk and never re-induced. This is AWM's own construction cost, distinct from SETA's zero-model-call EF rewrite (§G.10); Appendix S reports that asymmetry as a disclosed, not hidden, difference between the two methods.

```
System:
You convert a single successfully-completed multi-agent trajectory into a generalized,
     reusable WORKFLOW that could guide a DIFFERENT but similar future request in the same
     domain. This mirrors the workflow-induction step of Agent Workflow Memory (Wang et al.,
     2024).

Rules:
1. Output ONLY the numbered workflow steps, nothing else (no preamble, no closing remarks).
2. Each step names the tool or sub-agent invoked and its ROLE in the plan (e.g. 'Call
     get_employee_id to resolve the caller's identity'), not the literal argument values from
     this one trajectory.
3. Strip concrete entities: names, dates, ID numbers, amounts, emails. Replace them with a
     short placeholder describing what kind of value goes there (e.g. '<employee name>', '<
     leave date range>').
4. Preserve the ORDER of steps and any conditional branching you can infer from the trace (e.g
     . 'If the balance check fails, reject and explain why instead of proceeding to approval')
     .
5. Do not mention this specific trajectory's ID, user name, or exact figures anywhere in the
     output.
6. Keep the workflow to at most 12 steps.

User (template):
Domain: {usecase}
Original user request: {user_utterance}

Raw successful trajectory (tool calls in order):
{raw_trace}

Produce the generalized reusable workflow now.
```

`enable_thinking=false` is passed explicitly for this call, matching the NonThink $K{=}1$ deployment regime the AWM baseline is compute-matched against.

Injection format. Online, the top-1 retrieved TS record's cached induced workflow is rendered in the same retrieval slot SETA uses, with no EF or negative-polarity content (AWM has no failure-diagnostic mechanism; the formatter structurally reads only the TS bucket):

```
[Workflow retrieved from a prior successful case]

<induced, generalized workflow text for the top-1 TS record>

Adapt this general workflow to the specific details of the current
request; it is a plan template, not a literal transcript.
```

If no induced workflow is cached for the top-1 record, the formatter injects nothing rather than falling back to the raw trajectory text, so a cache miss cannot silently turn AWM into plain success-only retrieval.

G.12 Evaluation-time Final-Outcome Judge Prompt

This is a second, separate judge call from the outcome classifier c_ϕ of §G.3. It runs once per *evaluated online attempt* (not once per offline exploration trajectory), comparing the agent's own final message against the usecase's expected-outcome string, and its YES/NO answer becomes the `correct_final_outcome` field that this paper reports as final-outcome success throughout every results table. Appendix T already identifies the two judge-call types by code location and estimates their token cost; this is the first place either prompt's verbatim text is released. As with c_ϕ, the exact provider and model version are anonymized for review (Appendix A.2); for the bf16 and FP8 stacks this call is pointed at the same open-weight judge used for c_ϕ on those stacks (Appendix S).

```
You are evaluating whether an agent's final response matches the expected outcome.

Expected outcome: "{expected}"

Agent's final response: "{actual}"

Does the agent's response REPRESENT THE SAME FINAL DECISION as the expected outcome?
- "Request approved" and "approved the leave" = same
- "Request rejected" and "denied" = same
- A response that mentions "rejected" only as one of the options/criteria but actually says "
    approved" = NOT same as "Request rejected"
- Focus on the agent's ACTUAL FINAL DECISION, not words mentioned in passing.

Answer with only "YES" or "NO".
```

Called at temperature 0 with a bounded, geometrically-growing retry budget; a truncated or unparseable response is retried, never silently converted into a false label, and remains explicitly unscored (`judge_unscored_count`, §G.15) if every retry fails. The illustrative examples inside the prompt ("Request approved" / "Request rejected") are RTO-specific; the same fixed prompt text scores CRR attempts too, comparing against CRR's own expected-outcome strings (§G.9).

G.13 Reflexion Reflection Prompt

Used by the Reflexion sanity check of Appendix N to write one verbal, second-person reflection per failed task, before the next trial. Issued once per failed task by a closed proprietary model, anonymized as elsewhere in this paper and distinct from the RTO/CRR backbone (Appendix A.1).

```
You are a self-reflection critic for an autonomous tool-using agent.
The agent just attempted a task and failed. Your job is to produce a 2-3 sentence
verbal reflection that the agent can use on its next attempt. Focus on:
- What decision went wrong (which tool call committed to a state before verifying).
- What the agent should have checked first.
- A concrete instruction for the next attempt.

DO NOT restate the task. DO NOT praise the agent. Be specific and short.

User request: {user_utterance}
Agent final message: {final_message}
Expected outcome: {ground_truth}

Reflection (2-3 sentences):
```

The resulting reflections are injected as a labeled block, "Past attempt reflections," prepended to `use_case_instructions` for the next trial, visible to every agent (orchestrator and sub-agents alike) rather than only the orchestrator. This differs from SETA's own memory injection (§G.1), which reaches only the orchestrator; it is a property of the Reflexion baseline's own design, not of the harness's retrieval layer.

G.14 Content-specificity Control Text (E2T)

Appendix S reports a content-specificity check under Think-mode deployment (labeled E2T there): holding retrieval and prompt slot structure fixed, only the injected diagnostic's content is varied, to

test whether the Think-mode gain needs a diagnostic that matches the current task. One arm, *generic*, substitutes the fixed, task-agnostic sentence below for SETA's own task-matched EF diagnostic, in the identical retrieval slot:

```
A previous attempt reported completion before required verification
steps were completed.
```

This sentence is fixed across every task and seed; it is deliberately outcome-neutral and mentions no task-specific fact. The paired control, *shuffle*, instead substitutes another task's own real EF diagnostic in the same slot; §G.5 already gives the rendering both controls share around the substituted sentence.

G.15 K=4 Majority-Vote Rule

Standard K=4 controls (Appendix A) record 4 attempts per task and may aggregate them by majority; this is that aggregation rule, as implemented in the harness (`agent_arch/run.py`, function `_write_outcomes`), not a prompt. For a task with K scored attempts:

- **Final-outcome correctness (majority):** true iff more than half of the K attempts have `correct_final_outcome` = `True` (§G.12). A tie, for example 2 of 4, counts as *not* correct: the rule is a strict majority, not a plurality.
- **Strict task success (majority):** the same strict-majority rule applied to the stricter `task_success` field (`correct_final_outcome` `AND` no missing or extra state-changing tool call `AND` exact tool arguments; Appendix N).
- **Majority final message:** the most frequent exact final-message string among the K attempts; ties are broken by first occurrence, not by any semantic criterion, so this field is reported for inspection and is not used to compute either majority metric above.
- An attempt whose judge call remains unscored after every retry (§G.12) counts as `correct_final_outcome` $\neq$ `True` for the majority vote; the unscored count is retained separately (`judge_unscored_count`) rather than dropped from the denominator.

The K=4 Think-mode no-memory majority-vote baseline reported in Appendix S and Appendix A (D7) applies this rule with K=4; the two M-MEMO-K1 headline configurations record exactly one attempt per task and so never invoke this vote.

H Qualitative Case Studies: SETA Recovers False Successes

This appendix presents three qualitative case studies on the RTO (`requesting_time_off`) domain where SETA M-MEMO-K1 reliably resolves a task on which the K=1 baseline produces a false success across multiple seeds. All three patterns are drawn from the per-task pass rates of the 5-seed SETA M-MEMO-K1 sweep (SETA succeeds on 3/4 evaluable seeds in each displayed case; the remaining seed is a missing or invalid run, not a failure — here "invalid" denotes an environment/logging failure before a judgeable final output, excluded consistently from both numerator and denominator) versus the K=4 baseline aggregated across the standard 5-seed set $\{42–46\}$ together with two additional baseline-only dispatch seeds used for failure-mode persistence checking, giving n_{base}=28 attempt-seed pairs per task (K=4 × 7 baseline seeds; the additional 2 seeds are baseline-only and not used for any reported headline number, only for FS-pattern persistence verification). Each case is a representative *false success* pattern: the baseline fluently reports success while the agent did not actually meet the eligibility/policy preconditions, whereas SETA's failure-aware retrieval prefix biases the agent to either correctly approve, correctly reject, or declare an explicit failure with reason.

Case 1: Vacation request without verifying conflicts. Task 45 (Robert Wilson, vacation from Feb 1st 2024).

- **Baseline persistence audit (0/28 successes):** The agent fluently outputs "*The leave approval process has been completed successfully. The employee Robert Wilson has been verified, checked for el…*" — a confident report that does not actually confirm the conflict-detection step ran or that the approval routing succeeded. Ground truth: the request requires either an explicit reject or an unable-to-process declaration.

- **SETA M-MEMO-K1 (3/4 successes):** The retrieval prefix surfaces a similar past false success with the diagnostic "*Why this failed (judge analysis): conflict detection was bypassed before* `route_for_approval`." The agent now executes the conflict-detection tool first, observes a conflict, and emits the correct reject decision via the `finish` tool.

Case 2: Sick leave with fabricated approval. Task 9 (Ally Lee, sick leave).

- **Baseline persistence audit (0/28):** "*The leave request has been successfully processed and approved. According to the directions, I must respond with either...*" — the model commits to a successful approval before any tool actually returned an approval decision; the trailing reference to the directions hints at a meta-instruction leak rather than a verified action.
- **SETA M-MEMO-K1 (3/4):** The retrieved EF exemplar shows the correct "*Unable to approve sick leave because eligibility check returned conflict*" pattern. The agent calls `detect_leave_conflicts`, receives a conflict, and finishes with an explicit reject rather than a fabricated approval.

Case 3: Cross-month unpaid leave with policy violation. Task 52 (Stephanie White, unpaid leave Feb 12th onward).

- **Baseline persistence audit (0/28):** Identical fluent-success failure pattern ("*The leave request has been successfully processed and approved...*"). Cross-month unpaid leave triggers an unwritten policy condition that the K=1 agent does not surface because no retrieval signal reminds it of the exception.
- **SETA M-MEMO-K1 (3/4):** Retrieval prefix includes a TS exemplar whose typed-argument skeleton uses `leave_type=STR:"unpaid"` and an FS pitfall whose diagnostic states "*cross-month unpaid leave required additional approval routing that was skipped*." The agent applies the additional routing step and produces the correct response.

Cross-pattern observation. All three baseline failures share a common surface form — a fluent "successfully processed and approved" declaration that does not match the underlying ground truth. This is precisely the *false-success* (FS) failure mode that SETA's $g(\tau_{\mathrm{FS}})$ transformation targets. The recovery mechanism in each case is the same: a retrieved EF exemplar plus the labeller's *Why this failed* diagnostic biases the K=1 agent toward either (i) executing the missing verification tool, or (ii) emitting an explicit failure declaration via `finish`, rather than fabricating a confident-but-wrong success report.

Reading the cases. These are not the strongest individual deltas (the full per-task win/loss decomposition appears in Appendix D); they are illustrative examples of the FS→EF transformation in action on tasks where the false-success mode dominates baseline behavior. We emphasize that no per-task selection was used to construct the SETA pool: the same pool $\mathcal{M}$ serves all 60 tasks under the mature-memory regime, and these recoveries are the natural product of failure-aware retrieval rather than per-case prompt engineering.

H.1 Reflexion Compute-Allocation Sanity Check

The retained Reflexion evidence is a five-seed RTO sanity check over dispatch seeds $\{50, 60, 61, 62, 63\}$, reported in Appendix N. Its raw scored attempts use `pass_k` $= 1$; the stored artifacts do not establish equal total compute or byte-identical serving conditions, so this is not a matched-compute competitor and we draw no comparative conclusion from it. A preliminary direct seed-99 check yielded RTO task success of $70.0 \rightarrow 71.7 \rightarrow 68.3\%$ over three trials and CRR task success of $66.0 \rightarrow 66.0 \rightarrow 72.0\%$; these values are retained for provenance only.

I τ-bench Cross-Benchmark: Full Setup and Numbers

This appendix expands on the τ-bench cross-benchmark probe summarised in Section 5.1. Synthesis transfer is exact ($100\,\%$ of n=10 FS cases): every flagged FS case rewrites to a well-formed EF memory. End-to-end lift instead follows FS-substrate density, not the rewrite step: the airline probe is

boundary-level and retail has 0 % FS substrate (parity within 95% CI, detailed below); we omit a CRR figure, since no clean contrast survives there (§K).

Setup. Same primary backbone as the AgentArch experiments (§A.1). We construct a per-domain offline trajectory pool by running K=4 Think-mode rollouts on each task; FS detection follows the AgentArch protocol (consensus-failure on the closed reference judge). The online inference protocol matches SETA's main configuration: K=1 NonThink with retrieval injection. Because τ-bench tasks are multi-turn dialogue with longer reasoning horizons, we additionally port AgentArch's two thinking-mode tools — `synthesize_collected_information` and `calculator` — into the τ-bench tool surface so the bare K=1 NonThink backbone has the same multi-step reasoning capacity available to AgentArch agents. The model itself runs with `enable_thinking=False` at the chat template level (verified via vLLM `chat_template_kwargs`); thinking is therefore a tool-driven external scratchpad, not a hidden chain-of-thought.

Pool composition. Airline pool: 50 tasks × K=4 Think rollouts. TS rate 31.6 %. FS count: 5 (10.0 % of pool). Retail pool: 50 tasks × K=4 Think rollouts. TS rate 0.0 %. FS count: 0. EF count: 31 (77.5 % of pool), but *every retail EF entry has an empty captured trajectory and a final_message of "[a4-timeout: agent did not return within 200 s]"* — the primary backbone (§A.1) hits the K=4 Think 200 s wall-clock cap on every retail task during pool construction, before the agent emits a usable tool-call sequence. The consensus judge labels these as EF (the agent did not fabricate success), but the trajectories themselves carry no skeleton extractable for retrieval. The retail finding is informative as a backbone-capability ceiling result: the pool-construction stage on τ-bench retail is below the bandwidth threshold for this backbone under K=4 Think, well before the SETA online stage gets to operate.

Online evaluation grid (n=30 × 3 seeds). Three conditions, both domains, three seeds (42, 43, 44):

Condition (K=1 NonThink + thinking tools)	Airline Success (%)	Retail Success (%)
Baseline (no retrieval, no SETA)	30.00 ± 6.67	48.89 ± 6.94
Retrieval-only (SETA pool, no g)	38.89 ± 3.85	50.00 ± 3.33
SETA M-MEMO ($+g$ transformation)	28.89 ± 1.92	50.00 ± 3.33

SETA M-MEMO extension (n=50 × 5 seeds). To rule out that the n=30 grid lacked statistical power for the SETA condition specifically, we re-ran SETA M-MEMO+think at n=50 tasks per seed across 5 seeds (42–46) per domain, giving an effective per-domain sample of 250 trajectories. *Sample-grid disclosure:* the retrieval-only and baseline conditions in this appendix were measured at the earlier n=30 × 3 grid; only the SETA M-MEMO row was re-measured at n=50 × 5. Apparent point differences across the two grids should therefore be read as boundary-condition diagnostics, not as paired accuracy contrasts. A same-grid retrieval-only n=50 × 5 re-run is feasible and is documented as a follow-up.

Domain	Mean Success (%)	Std (%)	Per-seed (%)
Airline	35.20	3.03	$\{34.0, 40.0, 36.0, 34.0, 32.0\}$
Retail	45.60	6.99	$\{50.0, 46.0, 34.0, 46.0, 52.0\}$

0-substrate guard (retail). An initial retail extension run produced 42.40 ± 7.40 %, slightly below the 48.89 % baseline+think mean. Diagnostic inspection revealed that `_build_retrieval_block` was emitting a 684-character placeholder prefix (``(no tool sequence captured)'' + ``(no false-success trajectory in pool)'') on the retail 0-FS pool, because the EF candidates had non-empty label counts but empty trajectories. We added a 0-substrate guard: when no EF candidate has a captured trajectory, `_build_retrieval_block` returns `None`, matching `retrieval_only`'s 0-TS fallback. The number reported above (45.60 ± 6.99 %) is the post-guard re-run; per-record verification confirms `wiki_prefix_chars=0` on all 5 seeds, so SETA on the retail 0-substrate pool now degrades cleanly to baseline+think behavior. The remaining $\sim$ 3 pp gap below baseline+think mean is within seed-level 95 % CI and is attributable to sampling variance (seed 44 = 34.0 % is a cold outlier; the four other seeds average 48.5 %).

Interpretation. On AgentArch the SETA M-MEMO regime delivers a $+13$ pp accuracy lift over the K=1 baseline. On τ-bench the picture sharpens at n=50 $\times$ 5 seeds: on airline SETA reaches $35.20 \pm 3.03\,\%$; because this is the later n=50$\times$5 SETA-only grid while retrieval-only is from the earlier n=30$\times$3 grid, we make no pointwise accuracy comparison across those rows. On retail SETA reaches $45.60 \pm 6.99\,\%$ with the 0-substrate guard active, statistically indistinguishable from baseline+think at $48.89 \pm 6.94\,\%$. The most parsimonious explanation, and the one we adopt, is that SETA's accuracy lift requires the offline trajectory pool to contain enough false successes to populate the EF memory at meaningful scale — true on AgentArch RTO/CRR (5 task-level FS-flagged tasks out of 60 tasks; equivalently 5 FS trajectories out of $60 \times 4 = 240$ K=4 rollouts, i.e. $2.1\,\%$ trajectory-level FS rate under the K=4 rollout pool, a task-level $8.3\,\%$ FS-task rate), at the boundary of measurability on τ-bench airline (5 FS-flagged tasks; pool TS rate $31.6\,\%$), and below the boundary on τ-bench retail (0 FS-flagged tasks; pool TS rate $0\,\%$ plus 0 captured trajectories in the K=4 Think pool we built at the 200-second wall-clock cap). The retail finding is methodologically informative in a precise way: when there is no usable substrate, SETA's retrieval block correctly returns no injection (the 0-substrate guard) and the deployed agent operates on the no-injection code path, identical in code-flow to the K=1 NonThink baseline + thinking tools — no accuracy lift, no accuracy degradation. The g transformation mechanism transfers (every flagged FS in airline produces a well-formed EF memory); the headline accuracy benefit depends on FS substrate density, substrate quality, and retrieval-prefix fit, which are themselves functions of the (backbone, benchmark) interaction rather than of SETA itself.

Methodological note on thinking tools. The thinking-tool port is necessary because the τ-bench task structure (multi-turn customer-service dialogue with policy reasoning) has effectively no K=1 NonThink success rate on this backbone without external reasoning capacity; a pure SETA vs. baseline contrast would compare two near-zero numbers and would be uninformative. Adding the thinking tools to all three conditions equally lifts every condition into a measurable regime and isolates the SETA-vs-retrieval channel as the comparison of interest.

Substrate-density follow-up (n=3 seeds). Two relaxations probe whether the boundary observed at airline-5-FS / retail-0-FS shifts under denser substrate. **(a) Airline-65-FS**: the same 50-task split with K=4 Think yields 65 FS via deeper resampling (13$\times$ the original 5 FS); SETA-vs-retrieval-only delta $= -3.33 \pm 3.06$ pp (n=3 seeds, per-seed $\{0, -6, -4\}$ pp), reproducing the prior -3.69 pp regression at richer substrate. The substrate boundary is reconfirmed — substrate density alone is necessary but not sufficient. **(b) Retail-46-FS**: relaxing the K=4 Think wall-clock cap from 200 s to 400 s per task yields 46 FS / 0 timeouts (vs. 0 FS / $100\,\%$ timeouts at 200 s). The prior "retail K=4 yields no substrate" framing is therefore a *wall-clock artifact, not a domain property*. With substrate now present, the SETA-vs-retrieval-only delta is -5.33 ± 7.57 pp (n=3 seeds, per-seed $\{+12, -10, -18\}$ pp); per-task analysis identifies retail tasks #2 and #8 as consistently hurt across all three seeds — a concrete failure mode where SF-warning injection misleads the agent on policy-reasoning tasks. Both findings refine the FS-substrate density framing: substrate is necessary but not sufficient. On τ-bench, the interaction between substrate quality, task-policy structure, and retrieval-prefix fit can leave EF injection neutral or harmful. We therefore treat τ-bench as a negative/boundary probe, not as evidence of external accuracy transfer. The load-bearing signal in this follow-up is the per-task consistency of retail tasks #2 and #8 across seeds, not the seed mean. Full artifacts in `supplement_p1_3_taubench_extended/` of the supplementary archive.

J RTO Failure-Family Decomposition

Summary. SETA's gain concentrates on recoverable near-miss families (per-attempt incidence): CRR wrong-final-answer, missing-tool-call, and wrong-tool-choice fall $24 \to 12$, $14 \to 8$, $18 \to 14\,\%$; RTO wrong-final-answer falls $26.83 \to 11.33\,\%$; irrecoverable families ($\leq 1\,\%$) barely move.

This appendix gives the full RTO failure-family analysis, mirroring the CRR table above. A natural concern is that the mechanism story above is shown only on the long-horizon CRR domain; we add the same decomposition on the headline RTO domain to test mechanism consistency across the two AgentArch use cases.

Methodology. We bucket each per-attempt failure into one of four mutually-exclusive categories using a precedence-ordered heuristic over the AgentArch `record_level.csv` columns (no LLM re-judging, all categories derivable from the released trajectory artifacts):

1. `missing_tool_call`: the attempt's `missing_tools` list is non-empty.
2. `wrong_final_answer` (RTO-adapted): the actual orchestrator path's inclusion of `leave_approval_agent` disagrees with the accepted expected path's, i.e. a wrong approve/reject decision despite otherwise-clean execution. This is the RTO analogue of CRR's "correct path, wrong final answer" bucket — without the RTO-adaptation, almost all RTO failures collapse into `wrong_tool_choice` and the mechanism is masked.
3. `wrong_tool_choice`: extraneous tool, wrong orchestrator, or wrong order.
4. `other`: residual.

Data. Baseline = per-attempt continuous from the $K{=}4$ sweep ($n{=}1200$ attempts; 240 attempts $\times$ 5 seeds, matching the anchor used for the K=1 baseline row in Table 1). Full SETA M-MEMO-K1 = the same 5-seed pool that produces the $85.00 \pm 5.27\,\%$ headline in §5.1 ($n{=}300$ attempts; 60 attempts $\times$ 5 seeds). Pooled task-success reproduces the paper's SETA M-MEMO-K1 headline exactly ($85.00\,\%$); the baseline aggregate matches the derived per-attempt anchor reported in App. K (Cell 1).

Results. The CRR breakdown is reported alongside the RTO decomposition so that the cross-domain pattern is visible without compressing the family names into a sentence in the main text.

Table 10: CRR failure-family decomposition (% over all attempts; values are the baseline and full SETA conditions summarized above).

Failure family	Baseline	Full SETA
`wrong_final_answer`	24 %	12 %
`missing_tool_call`	14 %	8 %
`wrong_tool_choice`	18 %	14 %

Table 11: RTO failure-family decomposition (60 tasks, 5 seeds, mature-memory; % over all attempts; remainder = successful attempts).

Failure family	Baseline	Full SETA	Δ
`wrong_final_answer` (correct path, wrong approve/reject)	26.83 %	11.33 %	-15.50 pp
`missing_tool_call`	0.33 %	0.33 %	0.00
`wrong_tool_choice`	1.00 %	3.33 %	$+2.33$ pp
`other` (residual)	0.00 %	0.00 %	0.00
Total failures (= 100 % − TS)	28.17 %	15.00 %	-13.17 pp

Raw counts (audit): baseline $322/4/12/0$ over $n{=}1200$ attempts; SETA $34/1/10/0$ over $n{=}300$ attempts. The full per-seed CSV is released as `supplement_b3_analysis/rto_failure_family_table.csv` in the supplementary archive (alongside `rto_failure_family_REPORT.md` and the producing Python script).

Reading. The drop is concentrated in `wrong_final_answer` (-15.50 pp), the recoverable-near-miss family that the EF retrieval channel can target; the irrecoverable failure families (each $\leq 1\,\%$ in baseline) are too small to move further. RTO's failure mix is shorter-horizon than CRR's — a higher fraction of the baseline budget is already TS, leaving less room for absolute reductions in the rare-failure tails — but the largest reductions still occur in the recoverable near-miss family, mechanism-consistent with the CRR pattern above. The small $+2.33$ pp uptick in `wrong_tool_choice` sits

within seed-level noise (per-seed range 1.67–$6.67\,\%$ on 60-attempt cells) and does not reverse the headline.

K RTO K=1 Direct Mechanism Ablation

Summary. RTO's clean mechanism pair is the retrieval-only-vs-EF contrast (Table 1's Original row): retrieval-only stays near baseline, while the deterministic FS→EF rewrite contributes $+16.00$ pp over raw-FS retrieval (cells and specificity caveat below). We omit a comparable CRR $K{=}4$ marginal: the only surviving CRR ablation is a low-evidence, directional $K{=}1$ design (below), not treated as a mechanism marginal (Limitations). The success-only RAG rerun (E1; no FS or EF) is in App. M.

This appendix complements the CRR mechanism ablation above with a direct 4-cell ablation on the RTO domain at the K=1 deployment regime, addressing the natural concern that the headline mechanism evidence is reported on the long-horizon CRR domain rather than the headline-domain RTO.

Methodology. We measure 4 cells on AgentArch RTO 60 tasks under the $K{=}1$ deployment regime (one attempt per task). Final-message think-trace markers (App. O) show Cells 2, 3, and 4 (the M-MEMO-K1 headline) ran in Think mode (94–$99\,\%$ of final messages, pooled); we label them accordingly below. Cell 1 is a descriptive anchor ($71.83 \pm 0.77\,\%$) computed from the $K{=}4$ sweep behind Table 1 (no $K{=}1$ raw data exists for this cell), not a separately re-run, mode-matched $K{=}1$ measurement; we do not label its mode, do not tabulate it in Table 12 below, and do not use it in the comparisons that follow. Cells 2 and 3 are fresh $K{=}1$ measurements run on seeds $\{42, 43, 44, 45, 46\}$ on the same mature-memory pool used for the headline (Pool 122, no leave-one-out).

Table 12: RTO $K{=}1$ direct mechanism ablation (60 tasks, mature-memory pool). Cells 2 and 3 are fresh in-seed-family $K{=}1$ measurements, run as configured Think mode and retrospectively supported by final-message think-trace markers rather than request-payload verification (App. O), on identical seeds $\{42, 43, 44, 45, 46\}$, identical mature-memory pool, and identical retrieval index as Table 1; the +EF marginal of interest is the in-seed-family Cell 3 vs. Cell 2 contrast ($+16.00$ pp), which serves as the in-family direct +EF anchor consistent with the $K{=}1$ mechanism, not a competing headline or an independent confirmation. A descriptive anchor derived from the $K{=}4$ sweep behind Table 1 (Cell 1 in the Methodology paragraph above; no $K{=}1$ raw data for this cell, mode not labeled) is not tabulated here and is not used in the comparisons below. Cell 4 is the previously reported, configured-Think M-MEMO-K1 deployment headline from a separate dispatch, included only to align this diagnostic table with the main-paper anchor.

Cell	Configuration	n seeds	Success (%) ± std	Source
2	Think K=1 + retrieval-only (no EF)	5	72.33 ± 3.03	fresh K=1, seeds {42–46}
3	Think K=1 + retrieval + EF (`ef_induction_enriched`)	5	$\mathbf{88.33 \pm 3.33}$	fresh K=1, seeds {42–46}
4	SETA M-MEMO-K1 (Think, headline, descriptive)	5	85.00 ± 5.27	paper's canonical number (headline dispatch; not an anchor)

Results.

Reading. The in-seed-family +EF marginal at K=1 is Cell 3 − Cell 2 = $+16.00$ pp (both cells configured Think, App. O) on the same $\{42$–$46\}$ seed set, with both cells measured on the same mature-memory pool. Cell 3 vs. Cell 4 is *not* a head-to-head comparison (Cell 4 is from a separate dispatch and is included only as a descriptive anchor; see Caveats below). Together with the Section J per-attempt failure-family decomposition (`wrong_final_answer` -15.50 pp on RTO under SETA), this provides direct RTO-domain mechanism evidence at the deployment regime.

bf16 reproduction. On the corrected harness's bf16 stack, at the paper's original weight precision, the same retrieval-only-to-SETA contrast reproduces at $+15.67$ pp (95% CI $[+11.33, +20.33]$, $p < 0.001$; Appendix S).

Cross-domain × K-mode marginal-gain matrix (compactly). Combining the four-way grid:

+EF marginal	K=1 deployment regime	K=4 majority reference
RTO (this Appendix, Cell 3 − Cell 2)	$+\mathbf{16.00}$ pp (n=5, in-family)	not separately reported (below)

The CRR K=1 ablation was in fact performed, but is reported here only as a directional observation rather than as a headline result. It is a separate, earlier run set from the five-arm CRR study (Appendix W), so its numbers differ; the memory-placement results rest on the five-arm study. It used three cells (no-retrieval baseline, retrieval-only with TS+EFraw and no g, and full SETA with retrieval plus EFsynth/M-MEMO-K1), five seeds $\{42, 43, 44, 45, 46\}$, 50 tasks per seed, and `pass_k` $= 1$. Full SETA reached $67.6 \pm 2.33\%$, versus $69.6 \pm 0.8\%$ for retrieval-only and $68.8 \pm 1.6\%$ for baseline: a directional decrease of -2.0 pp relative to retrieval-only and -1.2 pp relative to baseline. Unlike the headline experiments, this execution used sampling temperature 0.7. The raw run artifact cannot be recovered; only the aggregate JSON remains, and no significance test (p-value, confidence interval, or bootstrap) was performed. We therefore treat these values as a low-evidence directional observation, not as a headline estimate or a significance claim. We do not additionally report a CRR K=4 mechanism marginal in the main text: an aggregate table from an earlier phase of this project records CRR values under Think K=4, retrieval, and EF conditions, but its retrieval-only-versus-EF contrast is not a purpose-built, unambiguous ablation pair the way the RTO Cell 2/Cell 3 design above is, no significance test isolates it, and no raw per-seed artifact survives; citing a specific marginal from it would overstate the precision the underlying data supports. The bootstrap-supported RTO K=1-versus-K=4 comparison above remains the within-domain mechanism reference, and the directional CRR K=1 observation just above is the paper's only CRR mechanism evidence.

Caveats. The clean mechanism contrast is Cell 3 vs. Cell 2: both are fresh K=1 measurements (configured Think, App. O) on the same seed set $\{42, 43, 44, 45, 46\}$ and the same mature-memory pool. Cell 4 is included only to align this diagnostic table with the main-paper anchor; it should not be used for additional significance claims, and it reproduces the published M-MEMO-K1 deployment headline from a separate dispatch as a descriptive anchor only. Cell 1 (the K=4-derived anchor discussed in the Methodology paragraph above) is likewise descriptive only and is not tabulated here. All cells, including Cell 1, are released in machine-readable form in the supplementary archive as `supplement_p1_1_rto_k1_ablation/AGGREGATE.csv`.

L Within-Family Stress Test (Smaller Backbone of the Same Family)

To complement the cross-family stress test in Appendix F (Family B), we also run a *within-family* stress test on a smaller-but-same-family backbone of the primary 35B-A3B model (call it Family A; the within-family variant is also a $\sim$ 35B-class A3B MoE model from Family A's design space, with comparable instruction-tuning style).[1] Within-family transfer probes a different boundary condition than cross-family transfer: the v2 induction template (Appendix G) was hand-authored against Family A's instruction-following style, so its prefix should remain compatible. The question is whether the SETA gain magnitude depends on the backbone's K=1 *baseline strength* — if the within-family baseline is stronger, the residual room for FS→EF amortization may shrink.

Setup. Identical configuration to the primary experiments: same RTO 60-task substrate, fresh offline pool from the within-family backbone's own K=4 Think-mode exploration trajectories (no primary-backbone pool reuse), same closed reference judge, same instruction-aware sentence encoder, same v2 enriched template, online K=1 inference. Five seeds $\{42$–$46\}$ aligned with the primary in-family replication seeds (App. K Cell 3) for paired comparability.

Reading the table. Three observations:

- **Sign-preserving transfer.** The SETA-vs-baseline contrast remains positive on the within-family backbone ($+2.67$ pp), and the retrieval-only-→-+EF ordering is preserved ($+4.67$ pp marginal of EF over plain retrieval, c.f. $+16.33$ pp on the primary model and $+2.22$ pp on Family B). The architecture- and template-level transfer are both intact within Family A.

[1] Specific backbone identifier held during anonymous review and disclosed at camera-ready.

Table 13: **Within-family stress test on RTO mature-memory (K=1, n=5 seeds $\{42$–$46\}$).** The within-family backbone has a substantially stronger K=1 baseline than Family A's primary 35B-A3B model. SETA preserves the gain direction ($\Delta > 0$) and the retrieval-only → +EF ordering. The table's primary-backbone +16.50 pp is the in-family $\{42$–$46\}$ measurement; the separately reported +13.17 pp is the headline dispatch's figure on $\{50, 60$–$63\}$, whereas the within-family variant is +2.67 pp on $\{42$–$46\}$.

Backbone	Base K=1	Retr-only K=1	SETA-v2 K=1	Δ vs base	Δ vs retr-only
Primary 35B-A3B (in-family $\{42$–$46\}$, App. K Cell 3)	anchor†	~ 72.00	$\mathbf{88.33 \pm 3.33}$	$\mathbf{+16.50}$	$\sim +16.33$
Within-family A-variant (n=5, $\{42$–$46\}$)	80.67 ± 3.59	78.67 ± 2.67	$\mathbf{83.33 \pm 1.83}$	$\mathbf{+2.67}$	$\mathbf{+4.67}$

† Descriptive K=1 baseline anchor (App. K Cell 1, derived per-attempt from the K=4 sweep; no K=1 raw data exists for this cell and its mode is not labeled); separately verified by a fresh n=5 re-run in App. M.

Pool FS rate (Family A′). The within-family K=4 Think-mode offline exploration pool (built fresh from the within-family backbone, no primary-backbone reuse) labels are: 195 `verified_optimal` (TS + EF positive polarity) / 45 `corrected` (FS, false-success after closed-judge re-grading) / 45 `failed` (FF), for a pool FS rate of $45/(195 + 45 + 45) = 15.79\,\%$. This is well within the primary backbone's 8–22 % pool FS range. The contraction from the headline dispatch (+13.17 pp on seeds $\{50, 60$–$63\}$) to the within-family marginal (+2.67 pp on seeds $\{42$–$46\}$) is therefore *not* explained by a thinner FS substrate; the offline pool has comparable failure-mode density to the primary backbone. The contraction is instead explained by the stronger online K=1 baseline (80.67 % vs. the primary backbone's derived K=1 anchor, App. K Cell 1), which mechanically shrinks the residual K=1 failure budget that FS→EF amortization can convert.

- **Magnitude shrinks with stronger baseline.** The within-family backbone's K=1 baseline is 80.67 %, vs. the primary model's derived K=1 anchor (App. K Cell 1). The residual room for FS→EF amortization is correspondingly tighter; SETA still lifts within that room, but to 83.33 % rather than 88.33 %. This is consistent with FS→EF acting on the *failure-mode budget* of the backbone — a stronger K=1 backbone enters production with a smaller FS+FF residual to convert.
- **Variance also shrinks.** The within-family seed-level pstdev drops from $\sim$ 3 pp (primary in-family) to 1.83 pp (within-family SETA), and from 5.27 pp (primary dispatch) further down — consistent with a stronger backbone resolving more tasks deterministically.

Boundary condition refined. Section 7 (External validity) identifies FS-substrate density and template-prefix-fit as boundary conditions for the FS→EF mechanism. The within-family result adds a third axis: *baseline-strength head-room*. The mechanism remains active and direction-preserving, but the magnitude of the headline lift is contingent on the backbone leaving enough K=1 failure budget for FS→EF to convert. This complements the Family B cross-family result (App. F) which isolates template-prefix-fit, and the τ-bench retail result (App. I) which isolates FS-substrate density. Together the three boundary axes circumscribe where the magnitude claim of Section 5.1 applies.

Methodological takeaway. The within-family transfer rules out a confound that Family B alone could not: that the magnitude shrink is purely a Family B (template-mismatch) artifact. Even within Family A, with a fully-compatible v2 template, the magnitude shrinks when the baseline is stronger. The contribution scope of this paper — FS→EF as a production-amortization architecture with a measurement protocol — is unchanged; the magnitude claim of Section 5.1 is descriptive of the primary backbone in particular and is explicitly not transposed across baseline-strength regimes without re-measurement.

M Vanilla RAG (TS-only) Baseline: Isolating the EF Transformation as the Active Component

The Section K mechanism table reports that, within the SETA pipeline, the *retrieval-only* ablation (which retrieves over the EF+neg pool but disables the EF transformation $g(\tau_{\mathrm{FS}})$) is at noise on RTO (+0.50 pp vs. baseline). A reviewer-level concern is whether the noise reading depends on the retrieval scope: SETA's retriever is configured to fetch from EF and neg channels in the ablation;

might a more conventional vanilla-RAG configuration — one that retrieves over a *success-only* memory of past TS trajectories — yield a different reading? We address this concern with a fresh $n{=}5$ controlled re-run.

Setup. We construct a separate baseline that retrieves over a TS-precedent memory only (success-only RAG): for each online query, the retriever returns the nearest task-instance trajectory whose offline label was TS, prepends its tool-call sequence as a few-shot exemplar, and disables both the EF and neg channels and the SETA $g(\tau_{\mathrm{FS}})$ transformation. All other configuration is identical to the primary experiments: same RTO 60-task substrate (and CRR 50-task substrate for cross-domain), $K{=}1$ online inference, fresh $n{=}5$ seeds $\{42\text{–}46\}$, same closed reference judge, same instruction-aware sentence encoder. The baseline holds for the existence of TS exemplars in the offline pool but pre-empts the FS detection and EF transformation that the SETA pipeline performs.

Table 14: **Vanilla RAG TS-only baseline ($K{=}1$ deployment, $n{=}5$ seeds $\{42\text{–}46\}$).** Both rows use the same offline pool, retriever, encoder, and closed reference judge as the SETA pipeline; the only difference is that the retriever exposes a success-only memory and the EF transformation is disabled. RTO baseline column repeats the App. K Cell 1 anchor for paired reading; CRR row reports an analogous anchor.

Domain	$K{=}1$ baseline (no retrieval)	Vanilla RAG TS-only $K{=}1$	Δ
RTO (60 tasks)	anchor†	71.33 ± 4.64	-0.50
CRR (50 tasks)	$\sim 70.00^{\ddagger}$	70.00 ± 0.00	~ 0

† App. K Cell 1 derived per-attempt anchor; like the pre-audit headline runs (App. O), this anchor's configured mode is not independently confirmed, so we do not treat this row as a mode-matched contrast. ‡ Analogous CRR anchor, same caveat; the 0-pstdev across all five seeds reflects deterministic majority outcomes on this domain at $K{=}1$.

Reading. Vanilla success-only RAG sits close to the RTO and CRR anchors used in Table 14 (-0.50 pp on RTO, ≈ 0 pp on CRR with all five seeds returning $70.00\,\%$), but both anchors carry the mode caveat noted above, so we do not read the RTO comparison by itself as a confirmed-mode contrast. A cleaner, mode-confirmed same-stack reading comes from the FP8 stack (App. S): Think-mode retrieval-only memory versus Think-mode no memory is likewise at noise, $+1.00$ pp (95% CI $[-2.00, +4.00]$, $p = 0.56$). Combined with Section K's $+0.50$ pp retrieval-only-EF+neg ablation, the picture is consistent across both readings: **retrieval alone — whether over a success-only memory or a no-EF version of SETA's own pool — is at noise level**; the gain has to come from the FS→EF transformation $g(\tau_{\mathrm{FS}})$. The full SETA M-MEMO-K1 lift on RTO ($85.00\,\%$ headline dispatch / $88.33\,\%$ in-family replication) is therefore not obtainable by inserting a vanilla RAG layer; the failure-mode-conversion pass is the active mechanism.

Anchor honesty. The body's $K{=}1$ baseline anchor of $71.83\,\%$ is derived (per-attempt anchor from the $K{=}4$ sweep, App. K Cell 1; no $K{=}1$ raw data exists for this cell and its mode is not confirmed) rather than re-measured, and it is not tabulated in the headline (App. K). A fresh $K{=}1$ re-run exists from the same pre-audit harness period, but because its own configured mode is subject to the same unverified-toggle risk documented in App. O and no per-record trace survives to audit it, we do not report it here as a mode-confirmed comparison point.

Why this matters. A common alternative hypothesis is that any retrieval mechanism — even success-only retrieval — could explain a deployment-time TS lift, because the retrieved exemplar nudges the agent's tool-call sequence toward known-good patterns. The data here directly contradict that hypothesis on two AgentArch domains under a fresh $n{=}5$ measurement: success-only RAG is $\Delta \approx 0$ vs. no-retrieval baseline. The FS-to-EF transformation g — which deterministically rewrites false-success trajectories into explicit-failure memory entries — is the load-bearing component. This is the precise sense in which SETA's contribution is the *selective failure memory*, not the retrieval scaffolding.

N REFLEXION MULTI-SEED SANITY CHECK

This appendix reports an evidence-limited, five-seed sanity check of Reflexion (Shinn et al., 2023) on the RTO split. It is *not* a compute-matched baseline: the non-empty artifact directory is `results/reflexion_cm_requesting_time_off_iclr2027/`; the former empty placeholder directory `results/reflexion_compute_matched_iclr2027/` was removed, and contained no result files. We therefore do not use the phrase "compute-matched" for this run or claim a matched-compute comparison.

What was run. The stored artifacts cover five dispatch seeds $\{50, 60, 61, 62, 63\}$ on the 60-task RTO split, using the same primary backbone described in Appendix A.1, temperature 0.3, a single serving endpoint and the open-weight judge used on the bf16 and FP8 stacks (not the original stack's closed judge), and a Reflexion trial/online configuration. The raw outcome records report `pass_k` $= 1$ for the scored attempts, not a $K{=}4$ majority. Each directory contains 60 scored task-level majority records plus the corresponding attempt records. Thus, the artifact supports a multi-seed RTO sanity check and latency instrumentation, but not the earlier claim that trial 1 was a $K{=}4$ compute-matched reference or that four passes formed a matched-compute grid.

Metrics. We report the same two fields defined in Appendix S: *final-outcome correctness* (the harness's `correct_final_outcome` field; the headline metric used throughout this paper) and *strict task success* (`task_success`), which additionally requires every needed tool call, no extra call to a state-changing tool, and fully correct tool arguments (`lenient_correct_tool_order` and `percent_of_correct_tool_args` $= 1.0$; `benchmarks/AgentArch/agent_arch/run.py:349--355`); despite the field's name, it does not check call order. For this revision we recomputed both fields directly from the `majority_outcome` rows of each seed's `outcomes.jsonl` (5 seeds $\times$ 60 tasks $=$ 300 records per configuration) and report final-outcome correctness as the headline column below, with strict task success retained as a secondary column rather than removed.

Table 15: **Stored Reflexion RTO sanity artifacts.** Five dispatch seeds, 60 tasks per seed. Percentages are mean $\pm$ sample standard deviation across seed-level rates; latency is the mean per-task wall clock across the five seed runs. These rows are descriptive and are not a matched-compute comparison with SETA.

Configuration	Final-outcome correctness (%) $\pm$ SD	Strict task success (%) $\pm$ SD	Latency (s/task)	Stored pass$_k$
Reflexion trial 1	76.00 ± 2.53	43.67 ± 6.71	45.16	1
Reflexion trial 2 (+ accumulated reflection)	68.00 ± 7.76	0.00 ± 0.00	98.70	1
Reflexion trial 3 (+ accumulated reflection)	62.67 ± 7.23	0.00 ± 0.00	99.56	1
Reflexion online $K{=}1$ (+ distilled lessons)	63.33 ± 5.00	0.00 ± 0.00	98.11	1

The outcome files contain 60 task-level majority records per seed and 300 records per configuration across the five seeds. We report the later-trial and online strict-task-success rows as stored, rather than replacing them with an unsupported aggregate. The latency files contain 60 per-task records per seed; they provide instrumentation for this sanity check, not a cache-normalized or byte-identical comparison against SETA. The retained request-audit records show the same backbone endpoint and temperature 0.3 across the configurations, but no retained artifact establishes equal total compute, equal context-window launch settings, or fairness with respect to the original SETA headline run.

Thinking-mode composition. These runs predate the corrected thinking-mode toggle: the request payloads do not record the thinking flag, and the fraction of final messages carrying model-generated reasoning text before a closing think tag is $43.0\,\%$ (trial 1), $38.7\,\%$ (trial 2), $47.3\,\%$ (trial 3), and $54.7\,\%$ (online $K{=}1$). The configurations therefore mix Think and non-thinking turns in uncontrolled proportions, unlike the original-stack headline runs (94–99 %; Appendix O) and the payload-verified bf16 and FP8 stacks.

Why strict task success reads $0.00\,\%$ for trials 2–3 and online $K{=}1$. Strict task success is a conjunction of final-outcome correctness, no missing or extra state-changing tool call, and exact tool-argument match (definition cited above; despite the field's name, it does not check call order). A run can report $0\,\%$ strict task success while final-outcome correctness stays well above chance

if either of the two additional conjuncts is never satisfied on that run. We checked whether the per-attempt `lenient_correct_tool_order` and `percent_of_correct_tool_args` values are independently recoverable for these runs. They are not: `outcomes.jsonl` stores only the derived boolean `task_success`, and no `record_level.csv` or other per-field raw dump is retained under `results/reflexion_cm_requesting_time_off_iclr2027/` for trial 1, trial 2, trial 3, or the online $K{=}1$ configuration. Trial 1 uses the identical scoring path and an unmodified usecase configuration. The later trials' usecase files are produced by the multi-trial wrapper (`scripts/run_reflexion_baseline.py:189`, function `write_reflexion_yaml`), which copies the full base usecase file, including its tool-order and tool-argument ground truth, and edits only the instructions text. A missing-ground-truth harness defect is therefore not the most economical reading, but we cannot confirm or exclude it, or any other specific account of the collapse, from the artifacts we retained. We report the strict-task-success collapse to exactly $0\,\%$ as **undetermined** rather than attributing it to a mechanism, and we do not use it to support any claim beyond what final-outcome correctness (above) already shows.

Paired significance test. The five dispatch seeds $\{50, 60, 61, 62, 63\}$ are shared across trial 1, trial 2, trial 3, and online $K{=}1$, so the final-outcome-correctness decline can be tested as a per-seed paired comparison rather than read only from the unpaired means. Recomputing per-seed rates directly from each condition's `outcomes.jsonl` majority rows and pairing by seed against trial 1: trial 2 gives a mean paired difference of -8.00 pp (95% CI $[-14.33, -1.67]$, seed-level bootstrap, 10,000 resamples; exact two-sided sign test $p = 0.625$, 1 seed favoring trial 2, 3 favoring trial 1, 1 tie); trial 3 gives -13.33 pp (95% CI $[-18.33, -8.33]$; sign test $p = 0.0625$, 5 of 5 seeds favoring trial 1); online $K{=}1$ gives -12.67 pp (95% CI $[-16.33, -9.00]$; sign test $p = 0.0625$, 5 of 5 seeds favoring trial 1). This test is **underpowered**: at $n = 5$ paired seeds, even a unanimous 5-of-5 sign split cannot cross the conventional 0.05 threshold under an exact sign test, whose floor is $p = 2 \times 0.5^5 = 0.0625$; the trial 3 and online-$K{=}1$ comparisons both reach exactly that floor. The bootstrap intervals for all three comparisons exclude zero, but we report the sign test alongside it rather than the more favorable of the two: with this few seeds, we treat the direction as better supported than any specific significance claim.

Interpretation. On the headline final-outcome-correctness metric, accuracy is lower once accumulated reflection context is present (trial 1: $76.00\,\%$; trial 2: $68.00\,\%$; trial 3: $62.67\,\%$; online $K{=}1$: $63.33\,\%$), a decline of roughly 13 to 14 points that is directionally consistent with reflection context not helping on this task family; the paired test above is underpowered at this seed count but agrees in direction on every comparison. Because the thinking-mode mixture also differs across these configurations (see above), the decline is not a clean effect of reflection context. This is a materially different picture from a reading based on the strict-task-success column alone, which reflects the metric-conjunction effect described above rather than a collapse to near-random final answers. Neither reading identifies a causal mechanism, and neither supports a general Reflexion performance claim: this is one RTO task family, one backbone, one judge, and one stored run protocol. CRR was not re-run in this artifact and no cross-domain conclusion is made.

N.1 A Controlled Replication (FP8-on-H100 Stack)

The check above predates the corrected thinking-mode toggle and mixes Think and non-thinking turns in uncontrolled proportions. We separately ran a payload-verified replication on the FP8-serving-on-H100 arm described in Appendix S (FP8-quantized serving on the H100×2/vLLM $0.23.0$ server, Think mode confirmed via the per-request payload, temperature 0.7), using the same RTO task set and the same five seeds $\{42\text{–}46\}$ as that arm's no-memory, retrieval-only, and SETA measurements, so all four conditions are directly paired.

Paired comparison. Pairing Reflexion's most expensive stage (online $K{=}1$, after three trials and distilled lessons) against the reference arms task-by-task-and-seed ($n = 300$ each): versus no memory, $+3.33$ pp (95% CI $[-4.67, +11.33]$, $p = 0.44$, not significant); versus retrieval only, $+0.33$ pp (95% CI $[-7.67, +8.33]$, $p = 0.97$, not significant); versus SETA, -13.00 pp (95% CI $[-20.00, -6.00]$, $p = 0.0004$: SETA is significantly higher). Internally, trial 1 versus online $K{=}1$ also shows no significant change ($+2.67$ pp, 95% CI $[-5.67, +10.67]$, $p = 0.55$). Three trials plus distilled online

Table 16: **Controlled Reflexion replication, FP8-on-H100 stack.** RTO, 60 tasks, seeds $\{42–46\}$, Think mode confirmed by request payload. Reference rows repeat the same-stack no-memory, retrieval-only, and SETA arms from Appendix S.

Configuration	Final-outcome success (%)	Strict task success (%)
Reflexion trial 1 (no reflection yet)	70.00 ± 2.64	60.67 ± 5.85
Reflexion trial 2 (+ accumulated reflection)	71.33 ± 3.98	0.00 ± 0.00
Reflexion trial 3 (+ accumulated reflection)	72.33 ± 3.46	0.00 ± 0.00
Reflexion online $K{=}1$ (+ distilled lessons)	72.67 ± 7.13	0.00 ± 0.00
Reference: no memory	69.33 ± 1.49	57.67 ± 5.84
Reference: retrieval only	72.33 ± 1.90	62.00 ± 2.98
Reference: SETA	85.67 ± 3.84	47.00 ± 3.80

lessons move headline accuracy in neither direction with any confidence, and land significantly below SETA.

Strict task success. As in the check above, strict task success is normal at trial 1 ($60.67\,\%$) and then reads exactly $0.00\,\%$ at every one of the 900 trial×seed×task records from trial 2 onward (3 stages $\times$ 5 seeds $\times$ 60 tasks), with no exceptions. We report Reflexion's final-outcome success only and do not compare its strict task success beyond trial 1: the collapse is exact and total in the same way it was in the earlier check above, and `outcomes.jsonl` again retains only the derived boolean, not the underlying tool-order and tool-argument fields, so the mechanism remains **undetermined** rather than attributable to a specific cause. The likely candidate remains a harness artifact under investigation, consistent with the reading above: the multi-trial wrapper that writes each later trial's task configuration copies the base task file, including its tool-order and tool-argument ground truth, rather than regenerating it.

Cost. Counting only the online $K{=}1$ stage, Reflexion uses 26.91 backbone calls and 115,687.7 prompt tokens per task, against SETA's 13.79 calls and 31,562.6 tokens on the same stack: roughly $2\times$ the calls and $3.7\times$ the tokens. Counting the full pipeline (three trials plus the online stage), Reflexion uses 99.90 calls and 376,054.5 tokens per task, roughly $7\times$ the calls and $12\times$ the tokens of SETA. These counts exclude the separate reflection-generation call that produces each trial's written reflection: it uses a different client than the audited orchestrator and sub-agent calls and is not captured in our request logs, though its *count* is recoverable as the number of failing tasks per trial (259 reflection calls across the five seeds). Because this cost is real but uncounted, the ratios above are a lower bound that favors Reflexion, not an overestimate of its cost.

Run-history disclosure. Earlier stages of this replication were served at lower concurrency (`max-num-seqs 8`) before a planned server restart raised it to 256; two seeds' (42 and 45) online-stage lessons were regenerated after that interruption, using the same prompt as the original attempt. This affects only latency comparability, which we do not report; it does not affect the accuracy or token-cost comparisons above.

Reconciling with the check above. The uncontrolled check earlier in this appendix (seeds $\{50, 60, 61, 62, 63\}$, temperature 0.3, Think fraction 38.7–54.7 %) showed a monotonic headline decline (trial 1 $76.00\,\%$ to online $63.33\,\%$), which that check's own paired test found directionally consistent across all five seeds but underpowered ($p = 0.0625$). This controlled replication, on an independent stack with Think mode confirmed at $100\,\%$ and using the same seeds as this paper's other precision-controlled comparisons, instead shows a mild, non-significant *increase* (trial 1 $70.00\,\%$ to online $72.67\,\%$, $p = 0.55$). The direction is not the same across the two stacks, so we do not read either one as evidence that accumulated reflection changes headline accuracy in a specific direction. The one pattern that reproduces identically on both independent stacks, different seeds, temperatures, and hardware, is the exact collapse of strict task success to $0\,\%$ from the second stage onward, which we continue to report as undetermined rather than mechanistically explained.

Summary. On this backbone, Reflexion's most expensive configuration (three trials plus online distilled lessons) is not distinguishable from no memory or from retrieval alone on final-outcome success, is significantly below SETA (-13.00 pp), and its measured cost, even undercounted by omitting reflection-generation calls, is several times SETA's. We report this as a self-reported cost-and-accuracy comparison, not a compute-matched one.

Earlier single-seed check. Before the stored five-seed artifact, a preliminary direct sanity check for seed 99 yielded RTO task success of $70.0 \rightarrow 71.7 \rightarrow 68.3\,\%$ and CRR task success of $66.0 \rightarrow 66.0 \rightarrow 72.0\,\%$ across three online $K{=}1$ trials. These values are retained for provenance only; they are not combined with the five-seed artifact and support no matched-compute or general-performance conclusion.

O Self-correction of Prior Description

An earlier description of this system was inaccurate in four respects. First, it described explicit-failure (EF) memories as products of an LLM re-prompt. The implemented construction instead deep-copies a false-success (FS) trace and deterministically rewrites its terminal finish message, the mirrored finish result where present, an optional trailing orchestrator summary, and the outcome/status fields, while assigning a derived trajectory identifier and recording provenance. No model call is made and no trajectory is added; the original FS file is left untouched. A 12-pilot LLM re-prompt attempt produced zero valid FS→EF transitions and is not the source of the reported EF pool.

Second, the online formatter does not inject one memory. The retriever selects $top_{TS} = 3$, $top_{EF} = 2$, and $top_{neg} = 2$ with MMR ($\lambda = 0.7$), while the formatter renders one TS skeleton, up to two negative-polarity trajectories, and one EF memory: four trajectories total. The extra selections are not rendered.

Third, raw false-success trajectories are not discarded. They remain in the store, pass the retriever's eligibility checks, and are rendered as known pitfalls with diagnostics. Accordingly, the retrieval-only versus retrieval-plus-EF ablation measures the contribution of the EF channel in a system that already injects raw false successes; it is not a comparison between EF and an absence of failure information.

Fourth, the headline dispatch and the RTO Cell 2/Cell 3 measurements (App. K) were configured with thinking disabled, but the original harness dropped the per-request thinking-mode toggle after the first call in a process, so they executed in Think mode. We judge this by the presence of model-generated reasoning text preceding a closing `</think>` tag in the final message: $288/300$ ($96.0\,\%$) of headline records contain a closing `</think>` tag and $283/300$ ($94.3\,\%$) contain non-empty reasoning text before it; Cell 2 and Cell 3 records carry matched, non-empty `<think>...</think>` pairs at $297/300$ ($99.0\,\%$) and $292/300$ ($97.3\,\%$) respectively; the supplement's original-stack records reproduce these counts. We therefore label them Think; latency measured under this configuration does not represent non-thinking deployment and is not reported. The FP8 stack corrects this toggle path and verifies the requested mode via the request payload on every call. This defect operates only in the direction of losing the non-thinking flag and reverting to the server's Think default, never the reverse, so an offline exploration run configured for Think mode (Appendix A) has no mechanism by which it could have silently executed as non-thinking.

These corrections are stated here plainly to make the implementation and manuscript description agree.

P Held-Out Four-Fold Evaluation (Inductive Cross-Task Pools)

Protocol. This check uses the E3 M0 configuration of Table 32 (NonThink, $K{=}1$, retrieval + EF, same backbone, serving, judge, and metric), changing only which trajectories are visible. The 60 RTO tasks are split into four folds of 15 by task index. For fold f, the memory pool is built exclusively from trajectories of the 45 tasks outside f, and only the 15 held-out tasks are scored; the union over the four folds scores every task exactly once with no same-task trajectory visible. Five seeds ($\{42\text{–}46\}$) are run; the same-task reference is the corresponding E3 M0 run of the same seed, so the comparison is seed-paired. Values are task-level majority strict task success, as in Table 32; $\pm$ is the population standard deviation over seeds, matching that table.

Table 17: **Held-out four-fold result (RTO, E3 M0 configuration).** Held-out: pool built from the other 45 tasks. Same-task: E3 M0 of the same seed (Table 32). Per-fold columns give correct held-out tasks out of 15; percentage columns are strict task success.

Seed	f0	f1	f2	f3	Held-out (%)	Same-task (%)	Δ (pp)
42	8	7	9	7	51.67	56.67	−5.00
43	9	7	9	8	55.00	55.00	0.00
44	7	7	10	8	53.33	56.67	−3.33
45	7	8	9	7	51.67	55.00	−3.33
46	8	7	10	7	53.33	55.00	−1.67
Mean ± SD					53.00 ± 1.25	55.67 ± 0.82	-2.67 ± 1.70

Result. Held-out accuracy is $53.00 \pm 1.25\,\%$ against the seed-paired same-task reference of $55.67 \pm 0.82\,\%$: a paired difference of -2.67 pp, seed-paired bootstrap 95% CI $[-4.00, -1.33]$ pp, paired t-test $p = 0.035$ ($n = 5$ seeds). Excluding seed 46 (see below) gives $52.92 \pm 1.38\,\%$ and the same sign on every remaining seed but one (seed 43, tie). This held-out NonThink control does not establish cross-task transfer and should not be read as evidence about the Think-mode headline effect.

Run disclosures. (i) The first attempt at seed 46, fold 1 was aborted after roughly one hour: on a task that was *not* in the fold's held-out set, the agent entered a repeated sub-agent delegation loop that ended in a Python recursion error, the client retried the model call nine of ten allowed times, and the host's job harness then killed the process for low memory. The partial evidence (47 judged rows, request audit, log) is archived with the results; folds 1–3 of seed 46 were rerun in full from a clean sidecar state, all completed normally with zero unscored tasks, and the loop did not recur. Fold 0 of seed 46 and all folds of seeds 42–45 ran once. (ii) Seed 46 alone produced server-side "maximum context length" (32,768-token) rejections that were retried by the client: on three non-held-out tasks across folds 0, 1, and 3, and on one held-out task (fold 3), which then returned an ordinary final message and was judged as a failure. Seeds 42–45 produced none; the same rejection class appears in the seed-46 E3 M1 log, so we attribute it to seed-46 sampling rather than to this protocol. The serving configuration (`max-model-len` 32,768) was unchanged for the whole sweep. (iii) Two pilot runs of seed 42, fold 0 used to validate the fold plumbing are excluded from the tally. (iv) The 15-cell provenance audit cannot identify a unique source run for seed 42 M1: both `main_tp2` and `main_tp2_recovery` produce 33/60. Per-run commit hashes were not recorded, so reproduction is limited to the seed, configuration, and payload sidecar rather than a commit-pinned checkout.

Q Trajectory Structure: EFSM Extraction, Divergence Analysis, and Structure-Aware Retrieval

This appendix extends Experiment 1 with an explicit model of *where* trajectories of the same task diverge. It reports (i) an offline extended-finite-state-machine (EFSM) extraction from the stored trajectory pool, (ii) a divergence-pair analysis and a re-indexing of the failure-family registry onto EFSM transitions, (iii) a structure-aware retrieval key evaluated under the held-out four-fold protocol of App. P, and (iv) a guard-explicit variant of the EF rewrite. Items (iii) and (iv) are follow-up checks; neither improves accuracy, and both are reported as null results.

EFSM extraction. Following the extended-finite-state-machine formulation of Cheng and Krishnakumar (1993), we map each stored trajectory to a path over abstract states. A state is the triple (tool name, abstracted arguments, abstracted result), and the guard on a transition is the abstracted result of the preceding call. The abstraction is deterministic and PII-free: scalar arguments become `present`/`empty` or a small category (a leave type is `valid`/`invalid`; a date string is `valid_date`/`date_present`; an escalation flag is `yes`/`no`); booleans keep `true`/`false`; lists and objects become `empty`/`nonempty`; the balance check reuses the inclusive ISO date span of the EF rewrite (App. A.6) and yields `balance>=`/`balance<` relative to the requested days. State identifiers are hashes of the triple; the figure legend maps them to $S_0, \ldots, S_{14}$. Records that could not be parsed were skipped (1,247 across both use cases) and are not counted anywhere below. The extraction reads the same stored pool as the main experiments and issues no model calls.

Table 18 gives the extracted machine sizes. RTO is compact (18 states, 50 transitions) and its guards are dominated by `result=other` and object/list emptiness; CRR is larger (45 states, 83 transitions) with the same guard families.

Table 18: **Extracted EFSM sizes and dominant guards.** Guards are counted over distinct transitions.

Use case	States	Transitions	Most frequent guards (count)
requesting_time_off	18	50	result=other (16), object=nonempty (10), list=empty (7), result=true (5), result=false (4)
customer_request_routing	45	83	object=nonempty (35), result=other (29), list=nonempty (12), list=empty (3)

Where trajectories diverge. For each task we form ordered pairs of trajectories with different outcomes and record the first EFSM step at which their state sequences differ. Counts are combinatorial (all pairs within a task) and therefore descriptive, not independent samples. Table 19 summarizes; Figure 6 shows the pruned RTO machine. Three observations matter for the paper.

1. **Divergence is early and not at the final answer.** The median first-divergence step is 3 for RTO and 9 for CRR. Pairs that agree on the whole path and differ only at the terminal `finish` state are 5.0 % of RTO TS-vs-negative pairs (84 of 1,672) and 2.7 % of CRR (40 of 1,458). The "same trajectory, different final answer" case is therefore rare; most negatives take a different branch well before finishing.
2. **The EF rewrite preserves the negative's path.** Because $g(\tau_{\mathrm{FS}}) \to \tau_{\mathrm{EF}}$ is a deterministic rewrite of the recorded trace (App. A.6), the multiset of EF tool paths equals the multiset of negative tool paths in RTO, and TS-vs-EF divergence is identical to TS-vs-negative divergence (1,672 pairs, median 3). EF-vs-negative pairs (485 in RTO, 2,916 in CRR) still have a positive median divergence because they pair *different* failures of the same task. This is the structural reason the tool-path key of the next paragraph cannot separate an EF record from a negative record: the information that distinguishes them lives in the explicit failure message, not in the path.
3. **Failure families sit on a few transitions.** Re-indexing the eight registered families onto the RTO/CRR machines locates the two structural families on single transitions: the 44 RTO trajectories of `structural_requesting_time_off` on $S_2 \to S_5$ under `result=true`, and the 69 CRR trajectories of `structural_customer_request_routing` on one transition under `result=other`. The six silent-failure families are singleton transitions (two of them share $S_1 \to S_0$ under `result=other`). The three states with the most TS-vs-EF divergence in RTO are the leave-type validation (S_2, 868 pairs), the initial identifier lookup (S_1, 260), and `finish` (S_8, 258); the last is both terminal and a mid-trajectory hub, because some negatives call `finish` and then continue.

Table 19: **EFSM divergence pairs by use case and outcome pair.** Count is the number of ordered trajectory pairs within a task; median is the first differing EFSM step. TS-vs-EF equals TS-vs-negative by construction of the rewrite.

Use case	Pair type	Count	Median divergence step
requesting_time_off	TS-vs-negative	1672	3
requesting_time_off	TS-vs-EF	1672	3
requesting_time_off	EF-vs-negative	485	3
customer_request_routing	TS-vs-negative	1458	9
customer_request_routing	TS-vs-EF	1458	9
customer_request_routing	EF-vs-negative	2916	9

Structure-aware retrieval under the held-out protocol. Given the divergence picture, a natural question is whether retrieving by *path* rather than by text helps when the pool contains no trajectory of

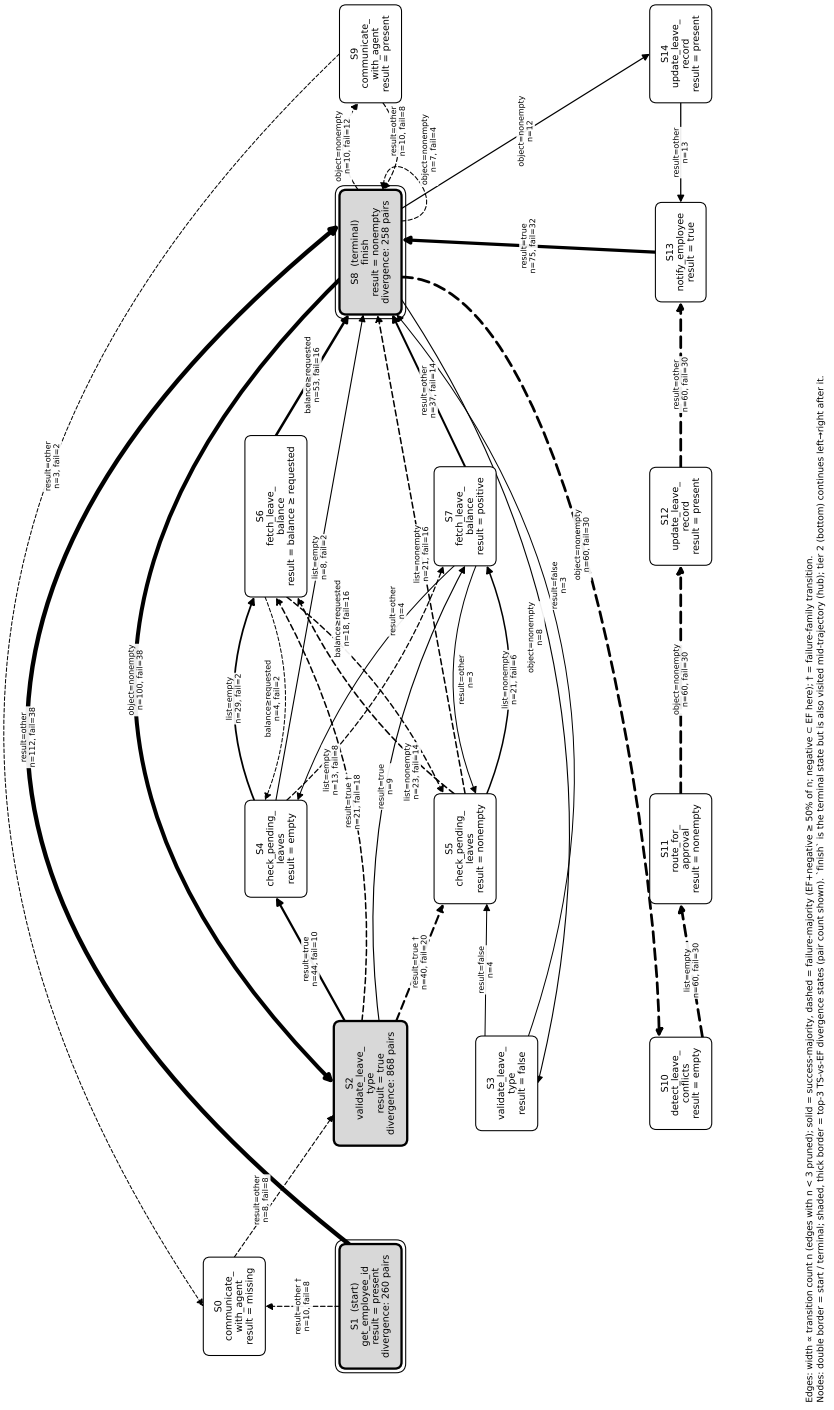


Figure 6: **Extracted RTO EFSM, pruned to transitions observed at least three times** (15 of 18 states, 34 of 50 transitions). Edge width is proportional to the transition count; dashed edges are failure-majority (EF plus negative trajectories are at least half of the count; negatives are a subset of EF by construction, so the two counts overlap); † marks transitions that carry a registered failure family; shaded nodes are the three states with the most TS-vs-EF divergence. `finish` appears both as the terminal state and as a mid-trajectory hub. State labels $S_0, \ldots, S_{14}$ follow the legend file released with the figure.

the query task. We add a retrieval key `path_lcs` to the three-way retriever: the top-1 TS record by text cosine is taken as the anchor, its abstract tool path is computed, and the EF and negative candidates are re-ranked by a hybrid score $0.5 \cdot \cos(q, c) + 0.5 \cdot \mathrm{LCS}(\pi_{\mathrm{anchor}}, \pi_c) / \max(|\pi_{\mathrm{anchor}}|, |\pi_c|)$, where LCS is the longest common subsequence of abstract states. Everything else (backbone, serving, judge, NonThink $K{=}1$, four folds, seeds 42–46, 15 held-out tasks per fold) is identical to App. P; the only change relative to that appendix is the retrieval key. Table 20 reports the result.

Table 20: **Held-out four-fold result with the structure-aware key (RTO).** Per-fold columns give correct held-out tasks out of 15; percentage columns are strict task success. Text: the text-cosine held-out result of Table 17 for the same seed. Same-task: E3 M0 of the same seed (Table 32).

Seed	f0	f1	f2	f3	Path key (%)	Text key (%)	Same-task (%)	Δ vs. text (pp)
42	8	7	9	7	51.67	51.67	56.67	0.00
43	9	8	6	9	53.33	55.00	55.00	-1.67
44	7	5	9	8	48.33	53.33	56.67	-5.00
45	7	6	7	8	46.67	51.67	55.00	-5.00
46	8	9	10	8	58.33	53.33	55.00	5.00
Mean $\pm$ SD					51.67 ± 4.56	53.00 ± 1.25	55.67 ± 0.82	-1.33

Result. Structure-aware retrieval does not recover the held-out loss. Over five seeds the path-key runs reach $51.67 \pm 4.56\%$ against $53.00 \pm 1.25\%$ for text retrieval under the identical protocol, a paired per-seed difference of -1.33 pp (95% bootstrap CI $[-4.33, +2.00]$, paired $t = -0.72$, $p = 0.512$, $n = 5$): on average indistinguishable from text retrieval but with markedly larger seed-to-seed variance, three seeds losing, one seed gaining five points (seed 46, 58.33%, the best cell of either protocol), and one unchanged. At the task level, 14 tasks flip to correct and 18 flip to wrong relative to the text-key run (paired $n = 300$). On point estimates the ordering remains path key $\leq$ text key $<$ in-pool ($55.67 \pm 0.82\%$), but the path-key gap to text retrieval is not statistically distinguishable. We report this as a negative result: the divergence structure explains *why* held-out pools lose accuracy (the failure signature is an early branch on a shared skeleton), but exploiting that structure at retrieval time does not recover accuracy, because in RTO every EF record shares the same abstract path as the successful run by construction, so the path key adds no discriminative signal and displaces text neighbours that carried task-specific detail; what it does add is sensitivity to which neighbours survive the displacement.

Run disclosures. All 20 folds (5 seeds $\times$ 4 folds, 15 tasks each) completed with exit code 0 and 15 evaluated tasks; a two-task smoke run preceding the sweep is excluded. In 3 folds the agent loop of at least one record hit the 32,768-token context limit and exited after the maximum iteration count (context-length errors per fold log: seed 44, fold 3: 84; seed 46, fold 1: 24; seed 46, fold 3: 10); the harness scores such a record as a failure and continues, which is the same behaviour observed in the text-key protocol (seed 46, folds 0/1/3: 16, 24, and 18 such errors). The sequential sweep launcher was terminated by the host for memory pressure while seed 46 fold 2 was initialising (no task of that fold had produced an outcome); its partial directory was set aside and folds 2–3 of seed 46 were relaunched from a fresh launcher with identical arguments. No completed fold was rerun.

Guard-explicit EF variant. The divergence analysis also suggests a more specific rendering of the EF message: instead of stating only that the trajectory failed, state the EFSM guard under which it left the successful path. We implemented this as a second deterministic rewrite variant (`--variant guard_explicit`) that looks up the failure family's transition and guard in the extracted machine and prepends a one-line, template-based statement of the violated guard (for example, that the leave-type validation returned `true` yet the request proceeded without the balance check). No language model is used at any point; where no family transition matches, the rewrite falls back to the canonical EF message. The mapping covers 17 of 23 RTO EF records directly (23 of 23 with the fallback), 47 of 52 CRR records, and both fragile-family records. The guard-explicit rendering was then evaluated in-pool under the same protocol as the in-pool control (Deployment NonThink K=1, five seeds, 60 tasks), with both arms scored on the same metric. On strict task success it reaches $54.00 \pm 3.74\%$ against $55.67 \pm 0.82\%$ for the canonical EF message (-1.67 pp, 95% bootstrap CI $[-5.33, +2.33]$, paired $t = -0.77$, $p = 0.48$, $n = 5$); on final-outcome correctness it reaches $64.67 \pm 3.06\%$ against

$65.00 \pm 1.05\%$ (-0.33 pp, CI $[-3.33, +2.67]$, $p = 0.86$). Naming the violated guard in the EF message therefore does not change accuracy at this sample size.

Limitations of this analysis. The abstraction is hand-specified per tool schema and would need to be re-specified for a new tool set; the pair counts are combinatorial and are not tested for significance; the EFSM is extracted from trajectories of the same 60 tasks and therefore describes this pool, not the task family in general; and the path key was evaluated only on RTO, where the EF and negative paths coincide by construction.

Reproduction. EFSM extraction, divergence pairs, and family re-indexing: `scripts/analyze_efsm_divergence.py --store-root results/ablation/trajectory_store --out-dir results/iclr2027_followup/t2_efsm`. Figure: `scripts/render_efsm_figure.py --efsm ... --divergence ... --min-count 3`. Structure-aware runs use the App. P launcher with `--retrieval-key path_lcs`; the guard-explicit store is produced by `scripts/synthesize_ef_bucket.py --variant guard_explicit`. Unit tests: `tests/seta/test_retriever_3way.py`.

R Integrated Ablation and Control Summary

The 36 ablation and control groups are summarized in Table 21. The table is an index, not a replacement for the detailed appendices cited in the "Raw artifact" column. We use the following evidence grades throughout: **A** means that a raw outcome file and a sidecar or manifest make the result traceable at cell level; **B** means that an aggregate artifact or statistical output exists, but the result is not traceable at the raw-record level; and **C** means that only a summary document is available. These grades describe traceability, not the direction or strength of a result.

Table 21: **Integrated summary of the 36 ablation and control groups.** "pp" denotes percentage points. A direction marked "inconclusive" or "mixed" is retained rather than collapsed into the headline result.

Experiment	Variable isolated	Design	Key result / Δ	CI / p	Raw artifact	Evidence	Direction
E2 C-GENERIC (Non-Think)	EF-substrate generalization / string length	RTO 23 + CRR 54 records/seed; seeds 42–46; $n = 5$; generic corpus	Seed accuracies 61.67, 63.33, 61.67, 66.67, 66.67%	CI crosses 0; $n = 5$ underpowered	`results/iclr2027_e2/`; corpus manifest and control raw	A	null / inconclusive
E2 C-SHUFFLE (Non-Think)	EF structural / order information	Same seeds/tasks as C-GENERIC; shuffled corpus	65.00, 66.67, 60.00, 68.33, 68.33%	CI crosses 0; specificity not established by sign/CI	Same E2 raw; `build_manifest.json`, logs, control records	A	null / inconclusive
E3 matched-K M0/M1/M2	Memory visibility versus K and think mode	15 cells; seeds 42–46; 60 tasks/cell; M0 K=1 Non-Think, M1 K=4 Non-Think, M2 K=4 Think	M0 55.67±0.82%; M1 54.00±2.00%; M2 32.67±2.26%; M0−M1 +1.67 pp	Paired block-bootstrap 95% CI $[-2, +5]$ pp; sign $p = .405$; 300 task-seed pairs	`results/iclr2027_e3/`; manifest, payload/outcome/retrieval sidecars	A	null
T1 held-out four-fold	Same-task visibility / cross-task transfer	4 folds × 15 RTO tasks; pool other 45; seeds 42–46; 300 outcomes	Held-out 53.00±1.25% vs matched M0 55.67±0.82%; $\Delta = -2.67$ pp	Paired t $p = .035$; bootstrap CI $[-4, -1.33]$ pp	`followup/t1_heldout/` (242 files)	A	negative

Continued on next page

Table 21 – continued from previous page

Experiment	Variable isolated	Design	Key result / Δ	CI / p	Raw artifact	Evidenc	Direction
T2 EFSM divergence	Where failure trajectories leave the successful path	RTO EFSM 18 states/50 transitions; CRR 45/83; divergence logs	RTO 1672/1672/485 pairs, median step 3; CRR 1458/1458/2916, median step 9	Descriptive; no p/CI	`followup/t2_efsm/` (209 files)	A	inconclusive (diagnostic)
T2 path-LCS held-out	Text retrieval key versus trajectory-structure key	Same held-out protocol; seeds 42–46	Path-LCS 51.67±4.56% vs text 53.00±1.25%; $\Delta = -1.33$ pp	CI $[-4.33, +2]$ pp; $p = .512$	`followup/t2_efsm_heldout/` (222 files)	A	negative / null
T2 guard-explicit in-pool	Retrieval key versus explicit EF guard rendering	In-pool; 5 seeds; canonical versus guard-explicit EF	Strict task success: guard-explicit 54.00±3.74% vs canonical 55.67±0.82%; $\Delta = -1.67$ pp (final-outcome: -0.33 pp)	$p = .48$ (final-outcome $p = .86$); CI $[-5.33, +2.33]$ pp	`followup/t2_efsm_guard_inpool/` (51 files)	A	null
RTO $K{=}1$ mechanism	Marginal retrieval and EF-rewrite contributions	4 cells; 60 RTO tasks × 5 seeds; $K{=}1$ Think	retrieval 72.33±3.03; +EF 88.33±3.33; headline 85.00±5.27%; EF +16.00 pp (Cell 1 anchor excluded, App. K)	Alternate bootstrap vs. a K=4-sweep-derived retrieval-only continuous anchor, not the fresh Cell 2 $K{=}1$ measurement: +16.17 pp; CI [9.17, 23.67] pp; $p{=}0$	`PAIRED_BOOTSTRAP_K1_CELL3.json`; raw CSV tree absent	B	positive (evidence caveat)
CRR $K{=}1$ mechanism	Whether the RTO EF effect transfers to CRR	3 cells; 5 seeds {42..46}; 50 tasks/seed; temperature .7	Full SETA 67.6±2.33%; retrieval-only 69.6±0.8%; baseline 68.8±1.6%; -2.0 pp vs. retrieval-only, -1.2 pp vs. baseline	No significance test performed; raw run artifact not recoverable	`PAT_B_CRR_K1_ABLATION.json`; aggregate only	B	negative (low-evidence directional observation; earlier run set, see App. W)
TS-only RAG	Success-only retrieval versus EF/SETA synthesis	5 seeds; RTO/CRR; $K{=}1$	RTO 71.33±4.64% ($\Delta = -0.50$ vs Cell 1 anchor); CRR 70.00±0.00%	CI/p unavailable	`PAT_D_RAG_TS_ONLY.json`; raw overall-scores tree absent	B	negative / null
Second backbone zero-shot	Backbone dependence	Family B 30B; $n = 3$ seeds 42–44; RTO/CRR; baseline/retrieval/SETA	65.56±1.57 / 64.44±2.83 / 66.67±3.60%; SETA +1.11 pp vs base	No reliable CI/p in cited summary	Summary JSONs; referenced raw trees absent	C/B	positive but weak

Continued on next page

Table 21 – continued from previous page

Experiment	Variable isolated	Design	Key result / Δ	CI / p	Raw artifact	Evidenc	Direction
Second backbone extended $n = 5$	Seed sensitivity of zero-shot result	Family B $n = 5$ extension	Base 67.00±3.80; retrieval 63.00±5.19; SETA 65.67±3.46%; SETA -1.33 pp vs base, $+2.67$ vs retrieval	No formal CI/p	Aggregate summary; raw provenance incomplete	C/B	negative vs base / mixed
Within-family variant	Same-family revalidation versus cross-family transfer	Family A′; $n = 5$; RTO; FS pool rate 15.79%	Base 80.67±3.59; retrieval 78.67±2.67; SETA 83.33±1.83%; $+2.67$ vs base, $+4.67$ vs retrieval	No formal CI/p	Aggregate summary JSON (Track B); raw tree absent	C/B	positive
Self-classifier replacement	External outcome classifier versus self-classifier/ routing	30 trajectories (TS:FS:EF = 10:10:10) × 5 seeds; $n = 150$	$\kappa = .727 \pm .078$; FS recall = 1.0; end-to-end pipeline -10 to -18 pp; adaptive top-k $+2.33$ pp	CI $[-1.67, +6.33]$ pp; $p = .26$	JSON aggregate/sweep artifacts; full-loop raw provenance incomplete	B	negative; mitigation inconclusive
Routing-fraction sweep	Whether fixed 25% hybrid routing is a routing-fraction artifact	0/10/25/40/55/ 100% fractions; pools not all identical	75.00±1.05 / 81.67±2.11 / 78.33±3.65 / 90.67±3.09 / 86.33±1.94 / 85.00±5.27%	No paired CI across all fractions; 25% uses different pool	`HYBRID_SELF_CLASSIFIER.json`; `HP_SWEEP.json`	B	mixed / inconclusive
τ-bench cross-benchmark	Benchmark transfer from AgentArch	Airline/retail; initial $n = 30 \times 3$; SETA-only $n = 50 \times 5$; K=4 Think pool, K=1 Non-Think+tools	Initial airline base/retrieval/SETA 30.00/38.89/28.89%; retail 48.89/50/50%; $n = 50$ SETA airline 35.20±3.03, retail 45.60±6.99; deeper SETA−retrieval airline -3.33 ± 3.06, retail -5.33 ± 7.57 pp	No matched paired CI; unpaired controls; placeholder correction documented	Summary and tau artifacts; no complete paired raw grid	C/B	negative / mixed
RTO failure-family decomposition	Which failure families account for the total failure change	Baseline $n = 1200$ attempts vs SETA $n = 300$; RTO/CRR registries	RTO wrong-final 26.83→11.33 (-15.50 pp; raw 322→34); missing .33→.33; wrong-tool 1.00→3.33 ($+2.33$); total 28.17→15.00%	Descriptive; unequal denominators; no CI/p	`results/ablation/trajectory_store/` (259 files), registry/CSV	A	positive overall; adverse subtype
FP8 stack, Think mode (D4a/D4b/ D4c/D5)	EF diagnostic vs. retrieval vs. no-memory vs. AWM, under Think	FP8 stack; RTO 60 tasks; K=1 Think; seeds {42–46}	No-mem (D4b) 72.67%; retr-only (D4c) 73.67%; SETA (D4a) 82.67%; AWM (D5) 71.00±3.46%	SETA−no-mem $+10.00$pp $[6.00, 14.00]$; SETA−AWM $+11.67$pp $[7.33, 16.33]$; AWM−no-mem -1.67pp (ns)	`results/iclr2027_fp8_diag_0925/`	A	positive (SETA only)

Continued on next page

Table 21 – continued from previous page

Experiment	Variable isolated	Design	Key result / Δ	CI / p	Raw artifact	Evidenc	Direction
bf16 stack, Think mode (P1/P2/P3)	EF diagnostic vs. retrieval vs. no-memory, under Think, at original weight precision	bf16 stack; RTO 60 tasks; K=1 Think; seeds {42–46}	No-mem (P1) 71.33±2.17%; retr-only (P3) 71.67±1.18%; SETA (P2) 87.33±5.48%	SETA−no-mem +16.00pp [11.33, 21.00]; SETA−retr-only +15.67pp [11.33, 20.33]; retr-only−no-mem +0.33pp (ns)	results/precision_study_0925/bf16/	A	positive (SETA only)
AWM matched comparison	Injected retrieval content: induced workflow (AWM) vs. EF diagnosis (SETA)	FP8 stack; RTO 60 tasks, K=1 NonThink; seeds {42–46}; AWM $n = 5$, SETA (EF-ref) $n = 5$	Final-outcome correctness: AWM 64.33±3.03% (n=5), SETA (EF-ref) 64.33±4.80% (n=5); paired $\Delta = 0.00$ pp (exact tie); strict task success paired $\Delta = +5.33$ pp (SETA favor)	Final-outcome: 95% CI [−4.00, +3.67], sign $p = 1.000$; strict: 95% CI [+0.67, +10.33], bootstrap $p = 0.032$, sign $p = 0.044$	supplement_awm_baseline/; App. S	A	null (indistinguishable, final-outcome); strict gap significant (SETA favor)
Precision control (Q1/Q2/Q3, FP8@H100)	Serving precision (bf16 vs. FP8), hardware/vLLM held fixed	FP8-quantized serving on the bf16 stack's own H100×2/vLLM 0.23.0 server; RTO 60 tasks; K=1 Think; seeds {42–46}	No-mem 69.33±1.49%; retr-only 72.33±1.90%; SETA 85.67±3.84%; precision DiD (bf16−FP8, same H100/vLLM) −0.33pp (full), +2.33pp (EF-only)	Precision DiD: [−7.00, +6.33]pp, $p = 0.970$ (full); [−4.00, +8.67]pp, $p = 0.500$ (EF-only); both null; hardware/vLLM DiD +6.33pp, $p = 0.054$, borderline	results/precision_study_0925/fp8_h100/	A	null (precision); hardware/vLLM effect borderline
E2T content specificity (Think mode)	Diagnostic content (task-matched vs. shuffled vs. generic), retrieval/slot fixed	FP8 stack; RTO 60 tasks; K=1 Think; seeds {42–46}; shuffle = other task's EF, generic = task-agnostic text	SETA 82.67%; shuffle 75.67%; generic 74.33%; retr-only 73.67%; SETA−shuffle +7.00pp; SETA−generic +8.33pp; shuffle/generic−retr-only +2.00/+0.67pp	SETA−shuffle [+2.67, +11.67]pp, p≈0.001; SETA−generic [+4.33, +12.67]pp, p<0.001; shuffle/generic vs. retr-only not significant ($p = 0.25$/0.74)	results/ic27_fp8_diag_0925/ (E2T arms)	A	positive (content-matched only)

Continued on next page

Table 21 – continued from previous page

Experiment	Variable isolated	Design	Key result / Δ	CI / p	Raw artifact	Evidence	Direction
D8 LLM-written lesson notebook	Audited EF diagnostic vs. LLM-generalized lesson, same retrieval slot	FP8 stack; RTO 60 tasks; K=1 Think online / NonThink lesson generation; seeds {42–46}; 23 lessons	D8 74.33% (strict 42.67%); SETA−D8 +8.33pp; D8−retr-only +0.67pp (ns); D8−no-mem +1.67pp (ns)	SETA−D8 [+4.33, +12.67]pp, p<0.0001; D8−retr-only [−2.33, +3.67]pp, p = 0.75; D8−no-mem [−1.00, +4.67]pp, p = 0.30	results/lesson_notebook/requesting_time_off/; App. S	A	null vs. retrieval-only; negative vs. SETA
Compute-amortization cost measurement	Online/offline backbone calls and prompt tokens, SETA (K=1) vs. K=4 majority vote	FP8 stack; RTO 60 tasks; K=1 SETA (D4a) vs. K=4 Think majority vote (D7, also used as C_{off} proxy); seeds {42–46}	SETA 14.10 calls / 31,939.6 prompt tokens per task; K=4 52.16 calls / 95,529.9 tokens; break-even $T^* \approx 1.4$ (range 1.36–1.50); SETA orchestrator-call prompt overhead +170.9% vs. no-mem (retrieval-only/ AWM $\approx$ +26%)	Descriptive (call/token counts, not a bootstrap comparison)	docs/iclr2027/0925_cost_measurement.json; App. A	B	positive (amortization favors SETA from the 2nd repeat query)
Reflexion 5-seed sanity	Iterative self-reflection as a SETA alternative	RTO 60 tasks; seeds 50, 60–63; temperature .3; pass@1	Final-outcome correctness (App. N): trial 1 76.00±2.53%, trial 2 68.00±7.76%, trial 3 62.67±7.23%, online K=1 63.33±5.00%; strict task success: trial 1 43.67±6.71%, trials 2/3/online 0.00% (cause undetermined); latency 45.16/98.70/99.56/ 98.11 s	No valid compute-matched head-to-head; descriptive only; paired per-seed test (App. N), trial 1 vs. online K=1: mean −12.67 pp, 95% CI [−16.33, −9.00], sign p = 0.0625; n = 5 seeds, underpowered	REFLEXION_COMPUTE_MATCHED_RTO.json + 100 raw files	A (descriptive)	negative (final-outcome decline); strict-success collapse undetermined
Identity metrics / failure recurrence	Whether SETA fixes baseline failures or creates regressions	RTO 60 tasks; three same-mode baselines, n=5 seeds each (App. D)	FRR / Update Yield / regressions, same-mode baseline vs. SETA — original stack (Cell 3 vs. Cell 2): 27.78% / 13 / 0; bf16 stack (P2 vs. P3): 35.29% / 11 / 0; FP8 stack (D4a vs. D4c): 68.75% / 5 / 1	Descriptive; no CI/p	Identity JSON/report summary; raw provenance incomplete	B/C	positive diagnostic

Continued on next page

Table 21 – continued from previous page

Experiment	Variable isolated	Design	Key result / Δ	CI / p	Raw artifact	Evidenc	Direction
Full-bootstrap multiplicity	Chance of multiple task/domain contrasts	$m = 56$; 28 pairs $\times$ 2; 10k/pair	Bonferroni survivors 16 (CRR 14/RTO 2), uncorrected 25, BH 21; RTO $p = .0002$ entries	Bootstrap + multiplicity correction; full per-cell table in App. E	Bootstrap summary/JSON; pair-level raw files not all recoverable	B	positive for survivors; otherwise inconclusive
Template reprompt pilot	Whether EF pool is LLM-reprompt synthesis	12 LLM reprompt pilots	Valid FS$\rightarrow$EF transitions 0	None	Pilot summary only	C	null / negative for synthesis claim
Memory placement, CRR (SETA-R/ SETA-S)	Placement of retrieved memory in long, multi-agent trajectories	bf16 stack; CRR 50 tasks; Think; seeds $\{42–46\}$; 5 arms $\times$ 5 seeds	No-mem 69.60%; SETA 69.60%; restated (SETA-R) 88.80%; per-step (SETA-S) 80.00%; SETA-R $+19.20$pp, SETA-S $+10.40$pp vs. no-mem	SETA-R CI $[+13.60, +25.20]$, $p < 0.001$; SETA-S CI $[+5.20, +16.00]$, $p < 0.001$; SETA vs. no-mem $p = 1.00$	App. W	A	positive (repeated placement only)
CRR content specificity, restated (2$\times$2)	Whether the restated-memory gain needs the EF diagnosis or only repetition	bf16 stack; CRR 50 tasks; Think; seeds $\{42–46\}$; content $\times$ restatement, 5 arms $\times$ 5 seeds	No-mem 69.60%; retr-once 69.20%; retr-restated 68.40%; SETA-once 69.60%; SETA-restated 88.80%; interaction $+20.00$pp	P2 (SETA-restated$-$retr-restated) CI $[+14.40, +26.40]$, $p < 0.001$; P3 (retr-restated$-$retr-once) CI $[-2.40, +0.80]$, ns; Holm sig. for P1/P2, not P3	App. W	A	positive (EF-specific; repetition alone null)
Memory placement, CRR, second stack (H100)	Whether the restated-memory gain is specific to the bf16-pro6000 stack	FP8 weights; H100 GPUs; CRR 50 tasks; Think; seeds $\{42–46\}$	No-mem 65.20%; SETA-restated 77.20%; $+12.00$pp (strict $+6.80$pp)	Headline CI $[+8.00, +16.40]$, $p < 0.001$; strict CI $[+3.60, +10.40]$, $p < 0.001$	App. W	A	positive (reproduces on 2nd stack)
Memory placement, RTO (SETA-S)	Per-step retrieval vs. single injection on shorter trajectories	FP8 stack; RTO 60 tasks; Think; seeds $\{42–46\}$	SETA-S 84.33% vs. SETA 82.67% vs. no-mem 72.67%; SETA-S$-$SETA $+1.67$pp; SETA-S$-$no-mem $+11.67$pp	SETA-S$-$SETA CI $[-3.33, +6.67]$, $p = 0.55$; SETA-S$-$no-mem CI $[+6.67, +16.67]$, $p < 0.001$	App. W.2	A	null vs. SETA; positive vs. no-memory

Continued on next page

Table 21 – continued from previous page

Experiment	Variable isolated	Design	Key result / Δ	CI / p	Raw artifact	Evidence	Direction
Memory placement, RTO (SETA-R restated)	Whether restating helps or hurts on shorter (RTO) trajectories	FP8 stack; RTO 60 tasks; Think; seeds {42–46}	SETA-R 86.00% vs. SETA 82.67% vs. no-mem 72.67% vs. retr-only 73.67%; SETA-R−SETA +3.33pp; SETA-R−no-mem +13.33pp; SETA-R−retr-only +12.33pp	SETA-R−SETA CI $[-1.33, +8.00]$, $p = 0.19$; SETA-R−no-mem CI $[+8.33, +18.67]$, $p < 0.001$; SETA-R−retr-only CI $[+7.33, +17.67]$, $p < 0.001$	App. W.2	A	null vs. SETA (single injection); positive vs. no-memory/ retrieval-only
Reflexion, controlled replication	Iterative self-reflection, payload-verified Think mode	FP8-on-H100 stack; RTO 60 tasks; seeds {42–46}; paired vs. same-stack no-mem/ retrieval/SETA	Online $K{=}1$ 72.67%; vs. SETA −13.00pp; vs. no-mem +3.33pp (n.s.); strict 0% from trial 2 onward (900/900)	vs. SETA CI $[-20.00, -6.00]$, $p = 0.0004$; vs. no-mem CI $[-4.67, +11.33]$, $p = 0.44$	App. N.1	A	negative vs. SETA; null vs. no-memory/ retrieval
Embedding/ instruction robustness	Embedding model size, instruction wording, and embedding family	FP8 stack; RTO 60 tasks; seeds {42–46}	0.6B SETA 80.00% (−2.67pp vs. 4B, n.s.); alt. instruction SETA 80.00% (−2.67pp vs. default, n.s.); e5-large-instruct SETA 80.67% (−2.00pp vs. 4B, n.s.); all three +6.33–7.00pp vs. retrieval-only-4B	0.6B CI $[-7.33, +2.00]$; instruction CI $[-7.67, +2.33]$, $p = 0.34$; e5 CI $[-6.67, +2.33]$; vs. retrieval-only $p = 0.006$/ < 0.001	App. X.1	A	null (embedding/ instruction/ family change); positive vs. retrieval-only
ReAct backbone pilot	Whether SETA's effect is specific to the function-calling orchestrator architecture	RTO; seed 42 only; ReAct-style agent	No-mem 11.67% (7/60); SETA 11.67% (7/60); exact tie	Pilot only, no significance test; ReAct architecture non-functional on this backbone (fabricates tool results)	App. X.2	C	inconclusive (broken backbone pilot)

The summary makes the asymmetry explicit: several positive mechanism or diagnostic results coexist with negative, null, mixed, and underpowered controls. In particular, the airline result on τ-bench, the Family B $n = 5$ sign flip, the self-classifier confidence interval, the increase in `wrong_tool_choice`, the directional decrease in the CRR $K{=}1$ ablation, the held-out and path-LCS decreases, and the absence of a cold-start masking improvement are not omitted from the evidence map.

On the same FP8 stack used in Appendix S, C-GENERIC and C-SHUFFLE were also run over a larger, non-overlapping seed range and remain close to the EF-reference arm on final-outcome correctness: C-GENERIC $64.45 \pm 3.97\%$ (15 seeds) and C-SHUFFLE $63.22 \pm 3.18\%$ (15 seeds), against $64.33 \pm 4.80\%$ (5 seeds, seed 45 now included) for EF-reference. These seed sets do not overlap with EF-reference's, so only the unpaired mean difference is available: $+0.12$ pp for C-GENERIC and -1.11 pp for C-SHUFFLE, both small relative to the EF-diagnostic marginal reported elsewhere in this paper. This NonThink-mode result cannot on its own establish content

non-specificity, however, since NonThink SETA is itself statistically indistinguishable from no memory (Appendix S): with no overall memory effect to explain, a null gap between EF-reference and content-swapped controls is uninformative about whether content matters. The Think-mode re-test of the same question (row "E2T content specificity (Think mode)" above) finds the opposite: EF-reference significantly exceeds both a shuffled and a generic control, and both controls fall back to the retrieval-only level (Appendix S).

S Matched Comparison Against a Workflow-Memory Baseline (AWM) and Mode Dependence of the Memory Effect

Purpose. A natural question about a memory-based agent method is how it compares with an existing training-free memory-construction baseline under matched conditions. This appendix reports a matched comparison between SETA and Agent Workflow Memory (AWM; Wang et al., 2024).

Setup. AWM induces reusable workflows from prior demonstrations via LLM prompting; we induced a workflow cache once from the same trajectory pool used elsewhere in this paper (103 induced workflows) and reused it unchanged across seeds, with no further re-induction. We compare this AWM configuration against SETA's EF-reference configuration on the FP8 stack: the same backbone under FP8-quantized serving and an open-weight outcome judge. Both arms share the same task set, trajectory pool, agent serving, and judge, and differ only in which content is injected at retrieval time (AWM's induced-workflow context versus SETA's EF-diagnosis context). The comparison uses RTO (60 tasks), seeds $\{42, 43, 44, 45, 46\}$, $K{=}1$ single attempt (NonThink), sampling temperature 0.7. SETA (EF-reference) seed 45 was initially missing (an embedding-service outage during the run prevented the retrieval context from being built); it was later re-run successfully against the recovered embedding endpoint, so both arms now report all five seeds.

Metrics. We report two outcome fields per seed. *Final-outcome correctness* (the harness's `correct_final_outcome` field) is the headline metric used throughout this paper: whether the judged final answer matches ground truth. *Strict task success* (`task_success`) additionally requires every needed tool call, no extra call to a state-changing (write) tool, and fully correct tool arguments; despite the name, it does not check call order (harness field `lenient_correct_tool_order`; §G.15). It is stricter and is reported for completeness, not as a replacement headline.

Table 22: **AWM versus SETA (EF-reference), matched FP8-stack comparison.** RTO, 60 tasks, $K{=}1$. Both AWM and SETA (EF-reference) use all five seeds $\{42–46\}$ ($n{=}5$ each; seed 45 for SETA was re-run after an initial embedding-service outage, see text).

	Final-outcome correctness (%)		Strict task success (%)	
Seed	AWM	SETA (EF-ref)	AWM	SETA (EF-ref)
42	61.67	60.00	41.67	53.33
43	63.33	68.33	50.00	56.67
44	68.33	58.33	51.67	48.33
45	66.67	66.67	46.67	53.33
46	61.67	68.33	51.67	56.67
Mean $\pm$ SD	64.33 ± 3.03 ($n{=}5$)	64.33 ± 4.80 ($n{=}5$)	48.33 ± 4.25 ($n{=}5$)	53.67 ± 3.42 ($n{=}5$)

Paired comparison. With both arms now at their full five seeds (300 task–seed pairs), a 10,000-resample paired bootstrap gives, for final-outcome correctness, $\Delta = 0.00$ pp (SETA $-$ AWM; exact tie; 95% CI $[-4.00, +3.67]$; exact two-sided sign test $p = 1.000$ over 17 SETA-favoring, 17 AWM-favoring, and 266 tied pairs). For strict task success, $\Delta = +5.33$ pp in SETA's favor (95% CI $[+0.67, +10.33]$; bootstrap two-sided $p = 0.032$; sign test $p = 0.044$ over 36 SETA-favoring, 20 AWM-favoring, and 244 tied pairs) — with the fifth seed added, this strict-task-success gap is now significant by both tests, where it previously was not ($n{=}4$-vs-5: $+5.00$ pp, 95% CI $[-0.83, +10.83]$).

Resampling unit. This bootstrap resamples task$\times$seed pairs independently and identically (block size $b = 1$) over the 300 shared task-seed pairs. This is a different resampling granularity from

the primary Table 1 headline bootstrap, which is task-paired and cross-seed: each task's value is aggregated across seeds first, and the resulting per-task deltas are resampled (Appendix A). With only five seeds available here, a seed-block bootstrap would leave too few blocks to be informative, so we use the finer-grained task×seed resampling as a documented convention; it does not model residual within-seed correlation, and a seed-block design could in principle give a wider interval. The final-outcome effect reported above is null, so a wider true interval would make "indistinguishable" a safer reading, not a false one; the strict-task-success effect is now significant at this resampling granularity, so a seed-block design (fewer, larger blocks) could in principle widen its interval enough to lose significance, and we do not claim the strict-task-success gap is robust to that alternative resampling choice.

Interpretation. On the headline final-outcome metric, AWM and SETA (EF-reference) are indistinguishable at this seed count: the point estimate is an exact tie and the confidence interval is wide relative to any plausible effect. The strict-metric gap favors SETA in direction, but the interval crosses zero, so it is not a significant difference. The comparison is between two memory designs and does not by itself measure either method's benefit over having no memory at all; it speaks only to AWM versus SETA under matched conditions. On the same FP8 stack under NonThink mode — where, as just noted, SETA's own EF-reference arm is not separable from no memory — the diagnostic controls C-GENERIC and C-SHUFFLE are also indistinguishable from the EF-reference arm (Appendix R); because NonThink SETA shows no overall effect to begin with, this NonThink-mode comparison cannot on its own test content specificity. The Think-mode re-test of the same question, where the memory effect is present, is reported below ("Content specificity under Think mode").

Construction contrast. The matched accuracy coexists with a different construction. AWM's induction step prompts a language model to abstract a workflow from prior demonstrations and stores that abstraction. SETA's rewrite is deterministic and model-call-free: it edits the terminal fields of a copied trajectory while preserving and linking to the original false-success path (§3.1). At matched accuracy on this benchmark, the two methods are qualitatively different constructions; this comparison does not establish that either construction is more accurate than the other in general.

Think-mode replication. The comparisons above use the agent's non-thinking (NonThink) mode. On the same FP8 stack (FP8-quantized serving, open-weight outcome judge, sampling temperature 0.7), we repeat the SETA-vs-no-memory contrast with the agent's thinking mode explicitly requested and confirmed via the per-request forwarded parameter (Appendix O), rather than inferred after the fact. Over 5 seeds $\{42–46\}$ (n=300 task–seed pairs), SETA's EF-reference memory raises final-outcome correctness from a $72.67\,\%$ no-memory baseline to $82.67\,\%$: $\Delta = +10.00$ pp (95% CI $[+6.00, +14.00]$, $p < 0.001$; exact two-sided sign test, 36 SETA-favoring, 6 no-memory-favoring, 258 tied pairs, $p < 0.001$). This is a clear, significant gain under Think-mode deployment, unlike the NonThink parity reported above.

Strict task-success trade-off under Think mode. The same Think-mode comparison shows strict task success falling from $55.67\,\%$ (no memory) to $42.33\,\%$ (SETA): $\Delta = -13.33$ pp (95% CI $[-19.00, -7.67]$, $p < 0.001$). This trade-off is specific to Think-mode deployment: under the NonThink comparison above (all n=5 seeds), SETA's final-outcome correctness was likewise statistically indistinguishable from the no-memory baseline ($+0.67$ pp, 95% CI $[-3.00, +4.33]$), and strict task success was exactly tied between SETA and no memory (0.00 pp, 95% CI $[-5.00, +5.00]$), while AWM's strict task success was significantly *lower* than no memory's (-5.33 pp, 95% CI $[-10.33, -0.33]$).

Mode-dependence. Two further arms on the FP8 stack under Think mode bear on whether the Think-mode gain is specific to the explicit-failure diagnostic or a broader property of retrieval or of memory reuse under Think-mode deployment. Think-mode retrieval-only memory (identical retrieval, no explicit-failure diagnostic) reaches $73.67\,\%$ final-outcome correctness (strict task success $50.33\,\%$) over the same 5 seeds. Relative to Think-mode no memory, retrieval alone gives $+1.00$ pp (95% CI $[-2.00, +4.00]$, $p = 0.56$, not significant); adding the explicit-failure diagnostic on top of identical retrieval (SETA − retrieval-only) gives a further $+9.00$ pp (95% CI $[+5.00, +13.33]$, $p < 0.001$; sign test 35 favoring, 8 against, 257 tied). On strict task success, the same explicit-failure addition costs -8.00 pp (95% CI $[-14.00, -2.00]$, $p = 0.011$), while retrieval alone costs -5.33 pp

(95% CI $[-10.67, 0.00]$, $p = 0.063$); this retrieval-alone strict-cost estimate is directional rather than established, since its interval touches zero. This reproduces, on the FP8 stack under Think-mode deployment, a pattern resembling the original stack's Cell 3-minus-Cell 2 contrast (Appendix K): under Think-mode single-attempt deployment the gain again appears concentrated in the explicit-failure diagnostic rather than in retrieval exposure alone, though this is a replication on the FP8 stack, not a re-run of the original cells. Think-mode AWM reaches $71.00 \pm 3.46\,\%$ final-outcome correctness (strict task success $41.33\,\%$) over the same 5 seeds: SETA exceeds AWM by $+11.67$ pp (95% CI $[+7.33, +16.33]$, $p < 0.001$; sign test 43 favoring, 8 against, 249 tied), while AWM itself is statistically indistinguishable from no memory (-1.67 pp, 95% CI $[-4.67, +1.33]$, $p = 0.34$). On strict task success, AWM falls -14.33 pp versus no memory (95% CI $[-20.00, -8.67]$, $p < 0.001$), similar in magnitude to SETA's -13.33 pp decline, while AWM vs. SETA on strict task success is not significant (-1.00 pp, 95% CI $[-6.33, +4.33]$). One candidate, unverified mechanism: if acting on the EF diagnostic sometimes inserts an extra verification tool call (Appendix H), this would require the reasoning capacity Think mode provides, and would also explain the strict-task-success cost, since an inserted call deviates from the expected tool sequence even when it corrects the final answer; we do not verify this mechanism directly.

Content specificity under Think mode (E2T). A separate question from mode-dependence is whether the Think-mode gain requires a diagnostic that matches the current task, or whether any text occupying the same retrieval slot would do. We hold retrieval and slot structure fixed and vary only the injected diagnostic's content: *shuffle* substitutes another task's explicit-failure diagnostic in the identical slot, and *generic* substitutes task-agnostic boilerplate text. Both controls otherwise match SETA's retrieval and prompt structure exactly. All arms use the FP8 stack, Think mode, seeds $\{42–46\}$ (n=300 task–seed pairs per comparison).

Table 23: **Content specificity under Think mode (E2T).** FP8 stack, RTO, K=1, seeds $\{42–46\}$.

Arm	Final-outcome correctness (%)	Strict task success (%)
SETA (task-matched EF)	82.67	42.33
Shuffle (other task's EF)	75.67	42.67
Generic (task-agnostic text)	74.33	39.00
Retrieval-only (no diagnostic)	73.67	50.33
No memory	72.67	55.67

SETA exceeds both content-swapped controls on final-outcome correctness: SETA $-$ shuffle is $+7.00$ pp (95% CI $[+2.67, +11.67]$, $p \approx 0.001$; 34 task–seed pairs favoring SETA, 13 favoring shuffle), and SETA $-$ generic is $+8.33$ pp (95% CI $[+4.33, +12.67]$, $p < 0.001$; 34 favoring, 9 against). Neither control is distinguishable from retrieval-only: shuffle $-$ retrieval-only is $+2.00$ pp (95% CI $[-1.00, +5.33]$, $p = 0.25$) and generic $-$ retrieval-only is $+0.67$ pp (95% CI $[-2.33, +3.67]$, $p = 0.74$). On strict task success, SETA and shuffle are indistinguishable (-0.33 pp, 95% CI $[-5.67, +5.33]$, $p = 0.95$), while both content-swapped controls cost strict compliance relative to retrieval-only (shuffle -7.67 pp, 95% CI $[-13.33, -2.00]$, $p = 0.012$; generic -11.33 pp, 95% CI $[-16.33, -6.00]$, $p < 0.001$) — the same qualitative strict-task-success cost pattern reported above for SETA itself. Under Think-mode deployment, only a diagnostic that matches the current task raises final-outcome correctness; substituting another task's diagnostic, or task-agnostic boilerplate text, in the same slot performs no better than retrieval alone. This resolves, for Think-mode deployment, the content-specificity question left open by the NonThink-mode check above.

LLM-written lesson-memory baseline (D8). We additionally compare SETA against a lesson-notebook baseline in the style of Mistake Notebook Learning (Su et al., 2025). For each audited explicit-failure pool item (23 items for RTO), the same backbone writes one short, generalized lesson (1–2 sentences) via a single NonThink call, conditioned on the task, the tool trace, the false final answer, the correct outcome, and the judge's stated reason for the failure (full prompt in Appendix G). Online, this lesson text replaces the EF diagnostic in the identical retrieval slot that SETA uses, with retrieval and slot structure otherwise unchanged; the online agent runs in Think mode, matching SETA's evaluation condition, while only the offline lesson-writing call is NonThink. FP8 stack, seeds $\{42–46\}$ (n=300 task–seed pairs).

Over five seeds, this baseline (D8) reaches $74.33\,\%$ final-outcome correctness ($73.33/75.00/76.67/73.33/73.33$ by seed) and $42.67\,\%$ strict task success. SETA exceeds D8 by $+8.33$ pp (95% CI $[+4.33, +12.67]$, $p < 0.0001$). D8 is statistically indistinguishable from both retrieval-only (D8 − retrieval-only $= +0.67$ pp, 95% CI $[-2.33, +3.67]$, $p = 0.75$) and no memory (D8 − no memory $= +1.67$ pp, 95% CI $[-1.00, +4.67]$, $p = 0.30$); on strict task success, D8 falls 7.67 pp below retrieval-only (95% CI $[-13.67, -2.00]$, $p = 0.008$) and 13.00 pp below no memory (95% CI $[-19.00, -7.33]$, $p < 0.0001$). A generalized, LLM-written lesson therefore performs at the level of raw retrieval, not at the level of SETA's audited, task-specific diagnostic: the gain reported throughout this paper requires the explicit-failure diagnostic's specific content, not merely a natural-language note occupying the same retrieval slot. This is consistent with the content-specificity result above (E2T): generic or mismatched diagnostic text in the same slot does not reproduce SETA's effect.

Reproduction on the bf16 stack. We additionally reproduce the Think-mode retrieval-only/SETA contrast on the bf16 stack (Appendix A): the same corrected harness, the same original bf16-precision backbone weights, the same offline memory pool, and the same open-weight judge as the FP8 stack above, but served on a 2-GPU server (tensor parallelism 2, vLLM 0.23.0) rather than the FP8 stack's single-GPU server. Over the same 5 seeds $\{42–46\}$, Think-mode no-memory reaches $71.33 \pm 2.17\,\%$, retrieval-only reaches $71.67 \pm 1.18\,\%$, and SETA (EF-reference) reaches $87.33 \pm 5.48\,\%$ final-outcome correctness. The EF-diagnostic marginal (SETA − retrieval-only) is $+15.67$ pp (95% CI $[+11.33, +20.33]$, $p < 0.001$); retrieval alone contributes $+0.33$ pp (95% CI $[-2.67, +3.33]$, $p = 0.91$, not significant); SETA − no-memory is $+16.00$ pp (95% CI $[+11.33, +21.00]$). Strict task success falls under SETA relative to no-memory by -10.00 pp (95% CI $[-15.33, -5.00]$), the same qualitative trade-off observed on the FP8 stack. This reproduces, at the paper's original weight precision and on the corrected harness, both the magnitude of the original stack's Cell 3−Cell 2 contrast ($+16.00$ pp, Appendix K) and the FP8 stack's finding that the gain concentrates in the explicit-failure diagnostic rather than in retrieval exposure alone.

Precision comparison, controlled (bf16 versus FP8, same hardware and vLLM version). The bf16-versus-FP8 comparison above leaves precision confounded with vLLM version, hardware, and tensor-parallelism degree: the bf16 stack runs on H100×2 (vLLM 0.23.0, tensor parallelism 2) and the FP8 stack on RTX PRO 6000×1 (vLLM 0.27.1, tensor parallelism 1). We separate these axes by running FP8-quantized serving on the bf16 stack's own H100×2 server with the same vLLM 0.23.0 (tensor parallelism 2 held fixed), rather than switching hardware and vLLM version at the same time. Under Think mode, seeds $\{42–46\}$ (n=300 task–seed pairs), this FP8-on-H100 arm reaches $69.33 \pm 1.49\,\%$ (no memory), $72.33 \pm 1.90\,\%$ (retrieval-only), and $85.67 \pm 3.84\,\%$ (SETA) final-outcome correctness: SETA − no-memory is $+16.33$ pp (95% CI $[+11.67, +21.00]$, $p < 0.001$), SETA − retrieval-only is $+13.33$ pp (95% CI $[+8.67, +18.00]$, $p < 0.001$), and retrieval alone is $+3.00$ pp (95% CI $[0.00, +6.33]$, $p = 0.076$, not significant) — the same qualitative pattern as bf16 and as the original FP8 stack above.

Holding hardware and vLLM version fixed at H100/vLLM 0.23.0, the *precision* effect (bf16 minus FP8) is -0.33 pp for the full SETA marginal (SETA − no-memory; 95% CI $[-7.00, +6.33]$, $p = 0.970$) and $+2.33$ pp for the EF-diagnostic-only marginal (SETA − retrieval-only; 95% CI $[-4.00, +8.67]$, $p = 0.500$): both are statistically null, with confidence intervals nearly symmetric around zero. Holding precision fixed at FP8, the *hardware-and-vLLM-version* effect (H100/vLLM 0.23.0 minus RTX PRO 6000/vLLM 0.27.1) is $+6.33$ pp for the full marginal (95% CI $[0.00, +12.67]$, $p = 0.054$) and $+4.33$ pp for the EF-diagnostic-only marginal (95% CI $[-2.00, +10.67]$, $p = 0.189$): the first is borderline by the conventional $\alpha = 0.05$ threshold and the second falls short of it, though both point in the same direction. These two decompositions are algebraically consistent with the uncontrolled bf16-vs-FP8 comparison above ($-0.33 + 6.33 = +6.00$ pp, matching the full-marginal figure there; $+2.33 + 4.33 = +6.66$ pp, matching the EF-diagnostic-only figure there within rounding).

Taken together, the originally observed bf16-vs-FP8 gap is attributable to hardware and vLLM version rather than to quantization precision itself: the precision effect is close to exactly null, while the hardware-and-vLLM effect accounts for essentially all of the original difference-in-differences. This experiment cannot further separate hardware and vLLM version from each other, however: tensor-parallelism degree (1 vs. 2), GPU generation (RTX PRO 6000 Blackwell vs. H100), and

vLLM version (0.27.1 vs. 0.23.0) all change together between the two servers, and each of the two decompositions above rests on only 5 seeds per arm, so these difference-in-differences carry wide confidence intervals and limited power.

Comparison against $K{=}4$ majority voting. On the FP8 stack in Think mode, we also ran a no-memory baseline with four attempts per task and majority voting over the judged outcomes ($K{=}4$, seeds {42–46}). Its final-outcome success is $72.67 \pm 1.90\,\%$, identical to the single-attempt no-memory baseline (0.00 pp, 95% CI $[-2.33, +2.33]$). A single SETA attempt exceeds it by $+10.00$ pp (95% CI $[+6.00, +14.00]$, $p < 0.001$, 300 task–seed pairs): four online attempts without memory buy no final-outcome gain, while one attempt with SETA memory does. On strict task success the $K{=}4$ baseline ($53.67\,\%$) stays above SETA ($+11.33$ pp, 95% CI $[+6.00, +16.67]$), the same strict-compliance cost reported above for retrieved memory.

Conclusion. Combined with the NonThink-mode parity reported above, the memory effect measured in this paper is specific to Think-mode single-attempt deployment: on RTO, in our setting, SETA's explicit-failure memory improved final-outcome success only when the agent ran in Think mode, not under NonThink deployment. We note that the Think-mode replications above use the bf16 and FP8 stacks rather than the paper's headline configuration — corrected-harness serving, an open-weight outcome judge, and sampling temperature 0.7, versus the original stack's closed reference judge and temperature 0 — so they are replications under different serving conditions, not a re-run of the original headline.

Machine-readable per-seed outputs for this comparison are released in the supplementary archive as `supplement_awm_baseline/`.

T Closed-Judge Cost and Data Exposure

Section 7 notes briefly that the closed reference judge is an external API; this appendix quantifies the cost and data-exposure implications that note does not. This appendix gives an order-of-magnitude estimate from existing logs and code, without any new judge or API calls. No retained artifact in the released logs stores actual per-call token-usage counts for either the judge or the backbone (`judge_audit.jsonl` has no usage field, `model_request_audit.jsonl` logs only forwarded request parameters, and no per-call token ledger is retained for the runs we inspected), so the token figures below are length-based estimates, marked as such, not measured billing. Provider identity and pricing tier are anonymized for review throughout, consistent with Appendix A.2.

Two judge-call types. The pipeline calls a judge in two distinct places, confirmed from `benchmarks/AgentArch/agent_arch/metrics.py`. (a) *Online per-attempt outcome scoring* determines `correct_final_outcome` for every scored attempt in every reported evaluation run, including the headline. The scoring function (`_get_correct_final_outcome`, line 436, calling `_llm_judge`, line 45) passes only the ground-truth final response and the agent's own final message to the judge; it does not pass the tool-call trajectory. (b) *Offline pool-construction classification* is the four-way TS/EF/SF/FF prompt in Appendix G.3 (`SF` is the prompt's own token for what this paper calls a false success, FS), used once per offline trajectory when building the EF memory pool. This call does receive the full trajectory. Both types can be routed to a closed reference judge; the headline results use type (a) throughout evaluation, and type (b) only at one-time pool construction per domain.

Call volume. Type (a) is one judge call per scored attempt: we confirmed this by matching `judge_audit.jsonl` line counts to attempt counts in the runs we inspected (a 60-task, $K{=}1$ run produces exactly 60 judge-audit records). At the deployed configuration ($K{=}1$), this is 1 call per task, so 1,000 deployed tasks require 1,000 judge calls; at the standard $K{=}4$ evaluation controls used elsewhere in this paper, it is 4 calls per task. The five-seed headline dispatch (Table 1) totals $60 \times 1 \times 5 = 300$ calls, matching its 255/300 and 85.00% denominator. Appendix A.2 already discloses a paper-wide empirical anchor for this call rate: 24,640 records across 112 runs, an average of about 220 calls per run, consistent with a mix of $K{=}1$ and $K{=}4$ runs at 60 scored tasks per run. Type (b) is a one-time cost per domain: $K_{\text{offline}} = 4$ trajectories per task gives 240 calls to build the RTO pool ($|\mathcal{M}| = 240$) and 200 calls for CRR ($|\mathcal{M}| = 200$); it does not recur per deployed query.

Tokens per call (estimated). For type (a), the fixed judge instruction is short (a bounded-retry yes/no comparison; we estimate 100–150 tokens), the ground-truth final response is short (observed `expected_length` in the tens of characters in the logs we inspected), and the agent's final message is typically one to a few sentences (observed `actual_length` on the order of 150–500 characters). Using a standard $\sim$4 characters-per-token heuristic for English text, this gives an estimated 145–305 input tokens per call; observed raw judge output in these logs is a single short token (e.g., "YES"), well under the harness's 128-token completion budget, so we estimate 1–10 output tokens per call. For type (b), the fixed classifier instruction in Appendix G.3 is longer: we measure 316 tokens for the exact `CLASSIFY_PROMPT` template text (the definitions, decision rule, and the five input-field placeholder lines, before substitution) using the public `o200k_base` tokenizer as a proxy for the undisclosed judge tokenizer (`cl100k_base` agrees exactly on this string, both giving 316 tokens); the dominant cost is the full Think-mode trajectory, and Appendix B already discloses that Think mode generates approximately 3K tokens per query, which we use as a proxy for trajectory size, i.e., a 2,800–3,200-token range around that figure. This gives an estimated 3,116–3,516 input tokens per call, with an estimated 15–45 output tokens (a label plus a one-sentence rationale).

Illustrative cost per 1,000 tasks. We do not know the actual provider or pricing tier used (anonymized for review), so we give only an order-of-magnitude estimate against a broad, explicitly illustrative price range spanning publicly listed prices for closed frontier-capable chat-completion APIs at the time of writing: roughly \$1–\$15 per million input tokens and \$3–\$75 per million output tokens. At $K{=}1$ (the deployed configuration), 1,000 tasks require 1,000 type-(a) calls, giving an estimated \$0.15–\$5.33 per 1,000 tasks. At the $K{=}4$ evaluation-grid rate used for the standard controls in this paper, the same 1,000 tasks require 4,000 calls, an estimated \$0.59–\$21.30 per 1,000 tasks. One-time pool construction is small by comparison: an estimated \$0.76–\$13.47 for the RTO pool and \$0.63–\$11.22 for the CRR pool, each paid once per domain rather than per query. These figures scale linearly with call volume and with whatever the actual per-token price turns out to be; they are meant to establish scale, not a specific budget.

Data exposure. The two call types expose different amounts of task content to a third-party API. Type (a) sends the ground-truth final response and the agent's own final message on every scored query; depending on the domain, a final message can restate task-specific details (for example, an HR leave-request decision that names an employee, or a customer-routing case that references account or contact details). Type (b) sends substantially more: the full user query, the complete tool-call trajectory (including whatever the invoked tools return), the agent's final answer, and the ground-truth answer, for every trajectory in the offline pool. In a real deployment, type (b) exposure is bounded and one-time per domain (a few hundred historical trajectories at adoption time), whereas type (a) exposure is smaller per call but recurs with every deployed query if an operator also uses the closed judge for ongoing production outcome monitoring; SETA's online $K{=}1$ path itself does not require a live judge call once the memory pool is built, so this recurring exposure is a monitoring choice, not a requirement of the deployed method.

Mitigation. Removing the closed judge entirely, by substituting a self-hosted, open-weight outcome classifier, eliminates third-party data exposure but is measured to cost an end-to-end pipeline regression of -10 to -18 pp relative to the closed-judge configuration (App. C). The same appendix reports a routing-fraction sweep across partial closed-judge usage, which recovers some of this gap at a reduced closed-judge call fraction rather than eliminating judge calls outright; the choice between these operating points is a deployment-time trade-off between data exposure, cost, and end-to-end accuracy that this paper does not resolve.

U Positioning Against Memory-Agent Methods

Table 24 places SETA next to the training-free or training-free-adjacent memory-agent methods named in §2, on axes that determine when each method's specific design choice matters rather than on a claim that any one method dominates. "Not established" means the cited description does not specify this axis and we did not read the underlying implementation to determine it; we report it as unknown rather than guess.

Table 24: **Positioning against memory-agent methods.** "Measured here" points to this paper's own comparison, where one exists; a blank comparison is not a claim that SETA is better, only that no compute-matched run against that method was performed in this paper.

Method	Memory content	Construction	Raw trajectory retained	Online attempts	Measured in this paper
SETA (ours)	Failure (audited EF), retained raw pitfall, success skeleton	Deterministic rewrite of an audited trace; no model call	Yes, provenance-linked	K=1	Table 1; all ablations
Reflexion (Shinn et al., 2023)	Reflection (verbal self-critique)	Model call (LLM writes the reflection)	Within-episode; not a persistent cross-task pool	Multiple trials, iterative	App. N (sanity check, not compute-matched)
Agent Workflow Memory (AWM; Wang et al., 2024)	Workflow (induced reusable procedure)	One-time offline model call induces the workflow from demonstrations	No; procedure abstraction, not the source trajectory	K=1 (our matched setup)	App. S (matched comparison; NonThink-mode parity, Think-mode SETA +11.67pp)
Mistake Notebook Learning (Su et al., 2025)	Failure (batch-clustered notes)	Not established from the cited description	Not established	Not established	App. S (our own operationalization, matched comparison; SETA +8.33pp)
Memento (Zhou et al., 2025)	Case-based (success/failure)	Learned case-selection policy from feedback	Case-based	Not established	Not measured here
ReasoningBank (Ouyang et al., 2025)	Reasoning strategy (LLM-synthesized, from success and failure)	Model call distills the strategy; optional extra test-time compute	No; abstracted strategy, not the source trace	K=1 retrieval, optionally more via extra compute	Not measured here
Dynamic Cheatsheet (Suzgun et al., 2025)	General strategy/code-snippet cache (task-agnostic)	Self-curated by the model each turn; model call	No; curated snippets, not the source trace	K=1	Not measured here
A-MEM (Xu et al., 2025)	Linked memory notes (Zettelkasten-style)	Model call writes each note at write time	No; note abstraction, not the source trace	Not established	Not measured here
Agent KB (Tang et al., 2025)	Cross-framework case plus diagnostic fix	KB aggregation with a learned disagreement gate	No; structured knowledge base, not the raw trace	K=1 per framework	Not measured here
AutoGuide (Fu et al., 2024)	Context-aware natural-language guideline	Offline model call generates and selects the guideline	No; guideline abstraction	K=1	Not measured here

The two rows with a compute-matched run in this paper (Reflexion, AWM) are each caveated at their own appendix: the Reflexion comparison is an RTO sanity check that does not establish equal total compute or byte-identical serving conditions (App. N), and the AWM comparison finds statistical parity with SETA's EF-reference arm under NonThink deployment (n=5-vs-5 seeds, $\Delta = 0.00$pp, 95% CI $[-4.00, +3.67]$ on final-outcome correctness) but a significant SETA advantage under Think-mode deployment (n=5-vs-5 seeds, $\Delta = +11.67$pp, 95% CI $[7.33, 16.33]$, $p < 0.001$; App. S). AWM's workflow cache is induced once, offline, by a separate LLM call over the same trajectory pool used elsewhere in this paper, then reused unchanged across seeds and across both modes; SETA's rewrite is model-call-free by contrast (App. O). "Compute-matched" throughout this table refers to matched online query-time compute, not total offline-plus-online compute. Mistake Notebook Learning and Memento are included for positioning only; no compute-matched comparison against either was run, nor against ReasoningBank, Dynamic Cheatsheet, A-MEM, Agent KB, or AutoGuide.

V Additional Details Moved from the Main Text

Outcome-pattern stratification (not semantic difficulty). Per-seed pass/fail patterns across the 60 RTO tasks are stratified rather than uniform, a seed-sensitivity pattern rather than a semantic-difficulty gradient. This stratification is observed only under the mature-memory regime (§3.4); we have no cold-start or cross-task data to test whether it holds elsewhere. Full counts: 37 all-passed (61.67%), 10 seed-62-only failures (16.67%), 11 mixed (18.33%), one seed-62-only pass, and one all-failed (1.67% each).

Cross-stack comparison caveat (Table 1). Raw final-outcome scores in Table 1 are not comparable across stacks: the original, bf16, and FP8 stacks differ in judge (closed vs. open-weight), precision, and sampling temperature, so only within-stack deltas (e.g., retrieval-only vs. retrieval+EF on the same stack) carry the reported confidence intervals and significance tests. The 88.33 % in-family

measurement is a secondary, different-seed-set figure and is not compared directly against the $85.00\,\%$ headline dispatch as a treatment effect.

W Memory Placement in Long Trajectories: Restated and Per-Step Retrieval

Motivation. The CRR $K{=}1$ ablation in Appendix K reports a directional, low-evidence result on an older stack. On the corrected bf16 stack, a single SETA injection on CRR is statistically indistinguishable from no memory (§5.4), unlike the clear RTO gain. CRR trajectories call the orchestrator many more times per task than RTO does; a diagnostic placed once, in the system prompt, is not visible at most of those later calls. This appendix tests whether repeating the memory near each decision recovers the effect, and reports the full cost and verification record behind §5.4.

Restated memory (SETA-R). At every orchestrator call, the same explicit-failure block that retrieval placed in the system prompt is appended again, verbatim, to the end of the input message. The system-prompt block is left unchanged, so the block appears twice after the first call. No new retrieval and no additional model call are added; only prompt tokens grow. Sub-agent calls are unmodified and never receive the repeated block.

Per-step retrieval (SETA-S). At every orchestrator call, we re-retrieve using the original request plus a length-capped summary of the trajectory so far (at most 6 turns, 800 characters), from the same candidate pool, embedding model, domain and outcome filters, and slot counts as the one-time retrieval. The freshly retrieved block *replaces* the previous call's reminder rather than accumulating with it. As with restated memory, the original system-prompt block is untouched and sub-agent calls never receive the repeated block. SETA-R and SETA-S therefore differ from each other only in whether the repeated block is fixed or freshly retrieved each time, and both differ from single-injection SETA only in repetition.

W.1 CRR Results

Table 25: **Memory placement on CRR.** bf16 stack, 50 tasks, Think mode, seeds $\{42\text{–}46\}$, $n = 250$ task–seed pairs per arm.

Arm	Final-outcome success (%)	Strict task success (%)
No memory	69.60	9.60
Retrieval only	69.20	8.00
SETA (single injection)	69.60	10.80
Restated memory (SETA-R)	**88.80**	10.40
Per-step retrieval (SETA-S)	80.00	8.80

A single SETA injection is statistically indistinguishable from no memory on CRR: $\Delta = +0.00$ pp, 95% CI $[-1.20, +1.20]$, $p = 1.00$ (1 win, 1 loss, 248 ties). Retrieval alone is likewise indistinguishable from no memory (-0.40 pp, 95% CI $[-2.00, +0.80]$, $p = 0.77$). Restating the same memory at every orchestrator call (SETA-R) reaches $88.80\,\%$: $+19.20$ pp over no memory (95% CI $[+13.60, +25.20]$, $p < 0.001$; 56 wins, 8 losses, 186 ties), and the same $+19.20$ pp over single-injection SETA, since single-injection SETA and no memory are themselves tied. Per-step retrieval (SETA-S) reaches $80.00\,\%$: $+10.40$ pp over no memory (95% CI $[+5.20, +16.00]$, $p < 0.001$; 38 wins, 12 losses, 200 ties). Strict task success stays near 8–11 % for every arm and does not improve under either placement.

Content specificity under restatement (2×2 control). Table 25's restated arm (SETA-R) repeats the EF diagnosis at every call. A separate 2×2 design crosses restatement (single injection vs. every call) with content (retrieval-only vs. retrieval+EF), to test whether the restated gain needs the EF diagnosis or only repetition.

Restating retrieval-only content, without the EF diagnosis, recovers only $68.40\,\%$: no better than single-injection retrieval-only (P3, not significant) and 20.40 pp below restated EF (P2, $p < 0.001$).

Table 26: **CRR 2×2: content $\times$ restatement.** bf16 stack, 50 tasks, Think mode, seeds $\{42–46\}$, $n = 250$ task–seed pairs per arm.

Arm	Final-outcome success (%)	Strict task success (%)
No memory	69.60	9.60
Retrieval only, single injection	69.20	8.00
Retrieval only, restated every call	68.40	12.40
SETA, single injection	69.60	10.80
SETA, restated every call (SETA-R)	**88.80**	10.40

Table 27: **CRR 2×2 contrasts (final-outcome success).** P1–P3 are pre-registered; Holm-Bonferroni corrects across the three. The interaction is a single test, reported uncorrected.

Comparison	Δ (pp)	95% CI	Holm result
P1: SETA-restated − no memory	+19.20	[+13.60, +25.20]	significant ($p < 0.001$)
P2: SETA-restated − retrieval-restated	+20.40	[+14.40, +26.40]	significant ($p < 0.001$)
P3: retrieval-restated − retrieval-once	−0.80	[−2.40, +0.80]	not significant ($p = 0.46$)
Interaction (restatement $\times$ content)	+20.00	[+14.00, +26.00]	$p < 0.001$ (uncorrected)

The restatement-by-content interaction is $+20.0$ pp ($p < 0.001$): restating helps only when the restated content is the EF diagnosis.

Second-stack replication (H100). To check whether the CRR restated-memory gain is specific to the bf16-pro6000 stack, we reproduce the no-memory-vs-restated-EF contrast on a second stack: FP8 weights on H100 GPUs, Think mode, the same judge and embedding model, all five pre-registered seeds.

Table 28: **CRR restated-memory gain on a second stack (H100).** FP8 weights, Think mode, 50 tasks, seeds $\{42–46\}$, $n = 250$ task–seed pairs per arm.

Arm	Final-outcome success (%)	Strict task success (%)
No memory	65.20	0.80
Restated memory (SETA-R)	**77.20**	7.60

On H100, restated memory reaches $77.20\,\%$: $+12.00$ pp over no memory (95% CI $[+8.00, +16.40]$, $p < 0.001$), where strict task success also rises, $+6.80$ pp (95% CI $[+3.60, +10.40]$, $p < 0.001$). The bf16-pro6000 stack instead shows this gain in final-outcome success only (Table 25). The two stacks are never pooled; this is a within-stack replication only.

Why the placement matters (task determinism). We tabulated each arm's five-seed outcome pattern per task. Without memory, 49 of 50 tasks (98.0 %) give the same outcome on all five seeds (34 always succeed, 15 always fail, 1 splits); SETA's pattern is identical, task for task, on 49 of 50 tasks (34/15/1); retrieval-only gives 48 of 50 (96.0 %; 33/15/2). Comparing SETA against no memory task by task, the five-seed pattern is exactly the same on 48 of 50 tasks (96.0 %); the two tasks that differ each flip on only one of five seeds, in opposite directions, so they cancel in the pooled comparison above. CRR success or failure is set almost entirely by which task it is, not by seed noise or by whether SETA's memory is present. This is consistent with the cost picture below: CRR trajectories average 38–41 backbone calls per task, of which the orchestrator accounts for only 6.0–6.9; single-injection SETA places its diagnostic in only the first of those orchestrator calls, so most of the trajectory, including every sub-agent call, never sees it, and the diagnosis's influence dilutes as the trajectory lengthens. Restating or re-retrieving the memory at every orchestrator call routes around exactly this dilution, which is why both placements recover a gain that single injection does not.

Memory block size. For the same seed-42 orchestrator request, the no-memory system prompt is 2,190 characters. SETA's one-time block is 13,826 characters (median), placed once. SETA-R appends 11,691 characters (median) to the same system prompt at every orchestrator call. SETA-S appends 9,737 characters (median) at every call, replacing the previous one. SETA and SETA-R therefore differ only in whether this block is restated; SETA-R and SETA-S differ only in whether the restated block is fixed or freshly retrieved.

Cost. Table 29 reports backbone calls and prompt tokens per task, split into the total and the orchestrator-only share.

Table 29: **CRR cost by memory placement.** Mean calls and prompt tokens per task; orchestrator columns isolate the calls that receive injected memory.

Arm	Calls/task (total)	Calls/task (orchestrator)	Prompt tokens/task (total)	Prompt tokens/task (orchestrator)
No memory	41.44	6.05	297,994.6	17,361.1
SETA (single injection)	40.90	6.24	310,785.0	37,767.0
Restated memory (SETA-R)	37.58	6.91	304,530.1	72,712.9
Per-step retrieval (SETA-S)	38.80	7.29	325,113.1	80,758.1

Total calls per task are similar across all four arms (37.6–41.4), and total prompt tokens are within 298k–325k. The difference concentrates in tokens per orchestrator call: relative to no memory, SETA's one-time injection raises orchestrator tokens 2.2× (17.4k→ 37.8k); restating it every call raises this to 4.2× (→ 72.7k); per-step retrieval raises it to 4.7× (→ 80.8k). Per-orchestrator-call tokens are more expensive under repetition than under one-time injection because a 10–12k-character block is added on roughly 6.9–7.3 calls instead of once. On this comparison, per-step retrieval is both more expensive and less accurate than restated memory: 11 % more orchestrator tokens (80.8k vs. 72.7k) and slightly more total calls (38.80 vs. 37.58), for a smaller gain (+10.40 vs. +19.20 pp, above). For CRR-length trajectories, restating the same memory is the cheaper and more accurate of the two placements; both clearly beat single-injection SETA, which is statistically inert on this domain.

W.2 RTO Results (Per-Step Retrieval Only)

Table 30: **Per-step and restated retrieval on RTO.** FP8 stack, 60 tasks, Think mode, seeds {42–46}, $n = 300$ task–seed pairs.

Arm	Final-outcome success (%)	Strict task success (%)
No memory	72.67	55.67
SETA (single injection)	82.67	42.33
Restated memory (SETA-R)	86.00	45.00
Per-step retrieval (SETA-S)	**84.33**	37.00

On RTO (60 tasks, shorter trajectories, orchestrator share 4.1–4.4 calls per task), per-step retrieval reaches 84.33 % final-outcome success ($n = 300$ task–seed pairs, seeds {42–46}, FP8 stack, Think mode): +1.67 pp over single-injection SETA (82.67 %; 95% CI $[-3.33, +6.67]$, $p = 0.55$, not significant; 31 wins, 26 losses, 243 ties) and +11.67 pp over no memory (72.67 %; 95% CI $[+6.67, +16.67]$, $p < 0.001$; 51 wins, 16 losses, 233 ties). Strict task success is 37.00 %, below single-injection SETA's 42.33 %. Per-step retrieval costs 2.7 % more calls per task than single-injection SETA (14.48 vs. 14.10) and 29.0 % more prompt tokens (41,207.2 vs. 31,939.6), for an accuracy gain that is not statistically significant. On RTO's shorter trajectories, repeated re-retrieval is not worth its added cost; the value of per-step retrieval in §W.1 appears specific to long, multi-agent trajectories such as CRR's.

Restated memory (SETA-R) reaches 86.00 % ($n = 300$ task–seed pairs): +3.33 pp over single-injection SETA (95% CI $[-1.33, +8.00]$, $p = 0.19$, not significant), +13.33 pp over no memory (95% CI $[+8.33, +18.67]$, $p < 0.001$), and +12.33 pp over retrieval-only (73.67 %; 95% CI $[+7.33, +17.67]$, $p < 0.001$). Strict task success is 45.00 %, not significantly different from single-injection SETA's 42.33 % (+2.67 pp). On RTO's shorter trajectories, restating neither helps nor hurts

significantly relative to a single injection: a single injection already reaches the decision, and restating does not significantly change final-outcome success on RTO.

W.3 VERIFICATION

The five CRR arms in Table 25 come from 25 runs (5 arms $\times$ 5 seeds). Every run scores 50 of 50 tasks, with zero unscored records, zero empty results, and the requested thinking flag confirmed on $100\,\%$ of requests via the per-request payload. Restated and per-step-retrieval reminders were confirmed, by direct inspection of the saved request payloads, to appear on every orchestrator call and on no sub-agent call, with no accumulation across calls (each call carries exactly one reminder).

W.4 EXCLUDED RUNS

Some runs behind this appendix were excluded before scoring, for process reasons unrelated to their outcome. We list them for transparency.

- **RTO restated-memory, first attempt (seed 42).** Generation completed ($60/60$ tasks), but the judge model configuration required for scoring was absent from the execution environment, leaving 53 of 60 records unscored; the run could not be rescored after the fact because the scoring pipeline requires the score file to exist before a later lookup step runs. The generated trajectories were still usable for the character-count measurements above, but not for accuracy.
- **RTO restated-memory, first attempt (seeds 43–46).** Stopped partway (12–20 of 60 tasks generated) once the seed-42 judge-configuration gap was discovered; continuing would have reproduced the same unscorable outcome.
- **RTO restated-memory, second attempt (seed 42).** Stopped before completion at the coordinator's direction, to avoid colliding with a planned server restart; preserved but unused, since it duplicated a generation already available from the first attempt.
- **RTO per-step retrieval, interrupted restart (seeds 42–46).** Killed intentionally, with approval, to restart the serving instance at higher concurrency; each had completed 0–1 of 60 tasks, so no scored data was lost. The completed reruns used elsewhere in this appendix replaced them.

The result of record for RTO restated memory and RTO per-step retrieval (both in Table 30, §W.2) is a single fresh regeneration for every seed, validated against five checks before use: 60 unique task identifiers; no unscored or null outcome rows; the thinking flag forwarded on $100\,\%$ of requests; a healthy reminder-block size (above a 1,500-character threshold, well clear of the roughly 800-character size an empty candidate pool produces); and, on failure, one clean retry before the run is reported as failed rather than retried indefinitely. During implementation, one candidate-pool path was passed as a relative path and resolved, inside a subprocess with a different working directory, to an empty directory; this produced an abnormally short reminder block (836 characters against a healthy 4,192) on the first live check, was traced and fixed by forcing an absolute path, covered by a regression test, and re-verified at a healthy block size (4,192–4,308 characters, within $3\,\%$ of the one-time block). A separate, empty result directory for CRR restated memory was also found at a path distinct from the one used for Table 25; it contained no data and was not used.

W.5 LIMITATIONS

Both placements are applied only at the orchestrator's own calls; sub-agent calls never receive the repeated or re-retrieved memory, so any gain from extending repetition to sub-agents is untested. Restating or re-retrieving memory does not address CRR's low orchestrator instruction-following rate, a separate issue neither placement targets. Costs are backbone calls and prompt tokens only; completion tokens and wall-clock latency are not measured, consistent with the cost accounting elsewhere in this paper (Appendix A).

X EMBEDDING AND BACKBONE-ARCHITECTURE ROBUSTNESS CHECKS

X.1 EMBEDDING MODEL AND INSTRUCTION ROBUSTNESS

Offline retrieval diagnostic. Before running any agent, we re-ran the retrieval step alone (no agent, no model call) on all 60 RTO queries against the same 257-item candidate pool used throughout this paper, comparing 6 embedding-model and instruction configurations. Domain-level accuracy (top-1 hit rate and retrieved-domain purity) is $100\,\%$ for all 6 configurations: the retriever's same-domain-first hard filter (§3.2) keeps retrieval from crossing domains whenever same-domain candidates are available, which they are throughout this pool, so domain accuracy alone cannot distinguish these configurations. The similarity *margin* between the top and next candidate does distinguish them: 5 of the 6 configurations (including the paper's default, Qwen3-Embedding-4B) have a margin of 0.36–0.53, while one variant not carried to agent-level evaluation below has a margin of only 0.16, roughly a third as wide, on this same pool. A thin margin does not itself change which task-domain is retrieved here, but it is a leading indicator of which configuration would be first to mis-rank within a harder or more balanced pool. Separately, removing the retrieval-time instruction text or replacing it with a different phrasing changes the *specific* trajectory selected within the correct bucket on 36–60 % of queries, even though bucket-level accuracy does not move: the instruction affects which memory is retrieved much more than it affects whether retrieval stays in-domain.

Agent-level results. We re-ran the RTO SETA condition end to end (FP8 stack, seeds $\{42$–$46\}$, $n = 300$ task–seed pairs, otherwise identical to the paper's main SETA configuration used throughout §5) with two single-axis changes from the paper's default (Qwen3-Embedding-4B with the paper's retrieval instruction):

Table 31: **Embedding and instruction robustness, RTO.** FP8 stack, seeds $\{42$–$46\}$, $n = 300$ task–seed pairs per comparison.

Configuration	Final-outcome success (%)	Strict task success (%)
Default (Qwen3-Embedding-4B) SETA	82.67	42.33
Smaller embedding (Qwen3-Embedding-0.6B) SETA	80.00	—
Alternate instruction (4B) SETA	80.00	36.33
Different embedding family (e5-large-instruct) SETA	80.67	38.67

With the embedding model alone shrunk to Qwen3-Embedding-0.6B, SETA reaches $80.00\,\%$: -2.67 pp relative to the default 4B configuration (95% CI $[-7.33, +2.00]$, not significant; 21 wins, 29 losses). With a different retrieval-time instruction (still 4B), SETA also reaches $80.00\,\%$: -2.67 pp relative to the default instruction (95% CI $[-7.67, +2.33]$, $p = 0.34$, not significant); strict task success is $36.33\,\%$. With the embedding model replaced by a different family (e5-large-instruct, the thinnest offline margin of the six candidates above at 0.16), SETA reaches $80.67\,\%$: -2.00 pp relative to the default 4B configuration (95% CI $[-6.67, +2.33]$, not significant) and $+7.00$ pp above retrieval-only-4B (95% CI $[+3.33, +11.00]$, $p < 0.001$); strict task success is $38.67\,\%$. All three variants remain clearly above retrieval-only with the default 4B embedding ($73.67\,\%$): the smaller embedding by $+6.33$ pp (95% CI $[+2.00, +11.00]$; 34 wins, 15 losses), the alternate instruction by $+6.33$ pp (95% CI $[+2.00, +10.67]$, $p = 0.006$), and the different embedding family by $+7.00$ pp above. These three comparisons each change both the embedding configuration and the presence of SETA's explicit-failure memory at once, so they are not a single-factor isolation of the embedding effect; we report them exactly as system-versus-system comparisons, not as an embedding-only ablation.

Interpretation. Across all three single-axis changes, SETA stays well above a retrieval-only system on the default embedding, and its own default-versus-changed difference is not statistically distinguishable from zero in any case. Smaller embeddings, different retrieval instructions, and a different embedding family change what gets retrieved (offline diagnostic above) without changing whether SETA's mechanism still helps. A third, weaker embedding configuration (e5-large-instruct, offline margin 0.16) shows the same pattern at agent level: indistinguishable from the default and above retrieval-only (above).

X.2 REACT BACKBONE PILOT

As a pilot check on a different agent architecture, we ran AgentArch's ReAct-style agent (reason-then-act, rather than this paper's function-calling orchestrator) on the 60 RTO tasks, seed 42 only. No-memory and SETA both reach $11.67\,\%$ $(7/60)$, an exact tie. On manual inspection, the ReAct architecture does not function correctly on this backbone: it fabricates tool results rather than issuing real tool calls, so the low, tied score reflects a broken backbone-architecture pairing rather than a measurement of SETA's effect. A five-seed run is in progress. We report this pilot to flag the negative result, not to draw any conclusion about SETA under ReAct-style agents.

Y MATCHED ATTEMPT-COUNT AUDIT (E3)

This appendix gives the full matched attempt-count audit summarized in the Results section: a self-contained check of whether one online attempt with visible memory sacrifices strict task success relative to four attempts without memory.

Y.1 MATCHED ATTEMPT-COUNT AUDIT

To separate attempt count from memory visibility, we ran a self-contained matched pair (E3) with the NonThink backbone, memory, retrieval/injection, serving, split, and judge fixed. The primary metric is task-level majority strict task success (final outcome correct, no missing or extra state-changing tool call, and arguments all correct, as defined in Appendix S) over 60 tasks and five seeds. This audit and App. P used the FP8 stack, so values are seed-paired internally but not level-comparable with Table 1. We label three configurations: M0 (NonThink, $K{=}1$), M1 (NonThink, $K{=}4$), and M2 (Think, $K{=}4$). M2 is excluded from the comparison below, since moving to Think mode also changes the maximum token budget.

With this configuration fixed, the internally seed-paired M0 and M1 means are summarized in Table 32. Across all 15 cells of this audit (E3), we verified payload purity, with zero missing, reversed, or cross-condition-contaminated cells; this audit covers only those 15 cells here and in App. P, since the headline dispatch and in-family runs behind Table 1 predate it and were not independently re-verified at the payload level. The table uses the FP8-stack E3 configuration (2-GPU TP=2, deterministic decoding, open-weight outcome judge, thinking disabled, seeds $\{42\text{–}46\}$); full rows and caveats are in Appendix P. This does not establish equivalence; it supports only the narrower choice that one online attempt need not sacrifice measured strict task success when the same mature memory is visible, under NonThink, where §5.3 finds memory produces no measurable final-outcome effect at all.

Table 32: **Matched attempt-count audit (E3), FP8 stack, NonThink deployment.** Task-level majority strict task success, mature-memory visibility, seeds $\{42\text{–}46\}$, $n = 300$ task–seed pairs. Paired 95% block-bootstrap CI $[-2.00, +5.00]$ pp; sign test $p = 0.405$. NonThink-only; does not establish $K{=}1$-vs-$K{=}4$ equivalence under Think mode (§5.3; Limitations).

Configuration	K	Strict task success (%) $\pm$ std
M0 (NonThink)	1	55.67 ± 0.82
M1 (NonThink)	4	54.00 ± 2.00