SYNACT: SYNERGISTIC ACTIVE FEEDBACK LEARNING FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents executing multi-step workflows fail in two fundamentally different ways: recoverable slips, which further attempts can repair, and knowledge gaps, which retrying cannot resolve. Expert feedback can address the latter, but expert attention is scarce. The key challenge is therefore not whether to ask for feedback, but *where feedback can change the outcome*. Existing approaches either avoid expert feedback, allowing agents to reinforce their own errors, or allocate feedback at the task level without identifying consequential decisions. We present SYNACT, an active learning framework that treats expert attention as the binding constraint and allocates feedback at the level of individual decisions. SynAct ranks decisions by the mutual information between the selected action and the terminal outcome across repeated rollouts, queries an expert at the highest-value decisions, and distills the resulting corrections into transferable rules that the agent can reuse without expert access at deployment. We theoretically characterize what this signal identifies under finite repeated sampling: with k rollouts and at least m observations per action, outcome-constant decisions have zero mutual information, while outcome-separating decisions have mutual information at least $H_2(m/k)$ bits, yielding exact separation for any threshold $t \in (0, H_2(m/k))$. This also explains why success-based acquisition can be blind when all rollouts fail: success is constant even when failed runs reach different outcomes. Across four agent benchmarks and four models spanning three families, SYNACT improves held-out accuracy in all 16 evaluated settings, by $+11.7$pp on average. The same signal transfers to inference-time routing, where SYNACT outperforms budget-matched cascade, uncertainty, and expert-judge routers in 15 of 16 settings, while a single targeted correction outperforms seven additional unaimed retries.

1 INTRODUCTION

An LLM agent can fail a multi-step task for two fundamentally different reasons: it can *slip* despite knowing how to succeed, or it can hit a *knowledge gap* that no amount of retrying will close. Additional test-time compute, through self-reflection (Shinn et al., 2023; Madaan et al., 2023) or trajectory search (Yao et al., 2023a; Zhou et al., 2024), can often recover the first but not the second: an agent checking its own work has little basis for distinguishing a correct interpretation from a confidently wrong one (Huang et al., 2024). An external expert can resolve such gaps, but expert attention is scarce. Agents that sidestep the expense by distilling context from their own traces (Zhang et al., 2026; Ouyang et al., 2026) return judgment to the same self-assessment that fails on knowledge gaps. In real agentic systems, the missing knowledge often resides in domain-specific procedures and human expertise (Liu et al., 2026a), making expert attention the one resource in the loop that does not scale. This creates a fundamental allocation problem: *when expert feedback is limited, where should it be spent?* A failing trajectory is not an atomic example: a 17-step episode may contain one or two decisions that determine its outcome and many that do not. Spending expert attention on the entire trajectory therefore wastes effort on irrelevant decisions. The right unit of feedback is not the trajectory, but the *decision point*.

We cast expert feedback as active learning over decision points. Rather than asking which trajectories are wrong, SynAct asks a more targeted question: *which action differences explain where the agent ends up?* We sample repeated rollouts of a task, group actions taken from the same environment state, and ask whether different actions lead to different terminal outcomes (Figure 1). We

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.

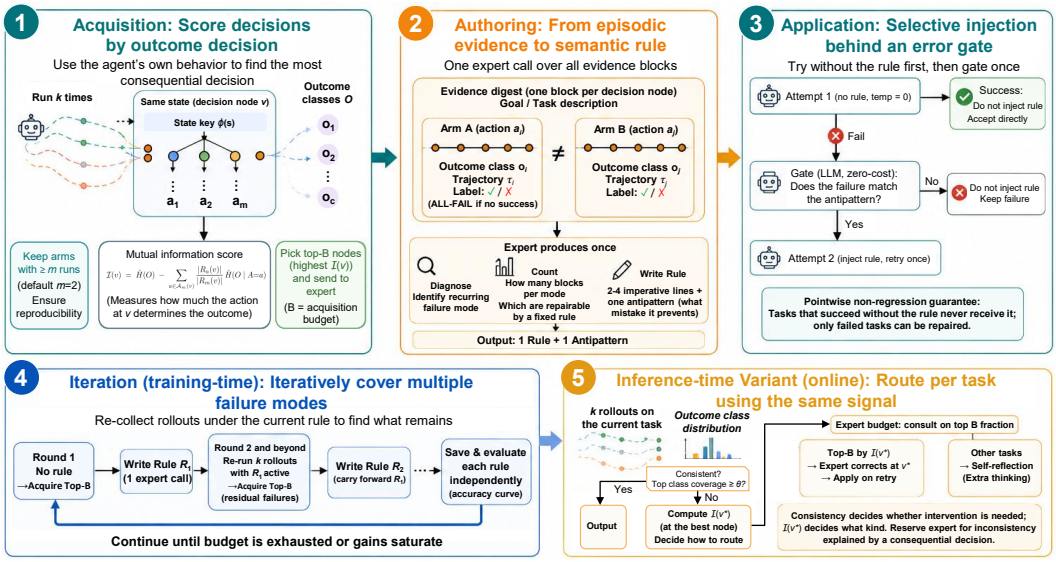


Figure 1: **SynAct allocates a fixed expert budget over decision points rather than episodes.** Acquisition scores each state-keyed decision node by $I(A;O)$, computed from the run itself with no gold answer or success check. Authoring sends the top-scoring nodes to the expert in one call, which abstracts them into a convention plus an antipattern. Application injects the convention only into failed attempts exhibiting that antipattern, iteration re-rolls to surface residual failures, and the same score routes consultation per task at inference time.

formalize this with the mutual information $I(A;O)$ between the action A taken at a decision point and the run's terminal outcome class O. Both are observable from the agent's own rollouts. Acquisition therefore requires neither a gold answer nor a success label. This gives the signal two properties that are particularly important for feedback allocation. First, it measures *consequential* disagreement rather than behavioral uncertainty: action entropy is high whenever runs disagree, whereas $I(A;O)$ is high when those action differences explain different outcomes. Second, it remains informative when every rollout fails. Different actions can still lead to different wrong end states, whereas a success-based signal collapses to zero when success is constant. SynAct can therefore surface decisions that matter precisely on tasks that are beyond the agent's current capability.

This raises a second question: once a consequential decision is found, how should its correction be turned into reusable knowledge? SynAct sends the selected decision points to the expert together and asks for a compact rule: a *convention* describing what to do and an *antipattern* describing the failure it prevents. At deployment, an error gate applies the rule only when a failed trajectory exhibits that antipattern. This separates three decisions that are often conflated: acquisition determines *where* expert attention is spent, authoring determines *what* is retained, and the gate determines *when* it is applied. The resulting rules outlive the trajectories that produced them, while the agent remains frozen and the expert stays out of the execution loop. We evaluate SynAct on four agent benchmarks spanning text-to-SQL, compositional crafting, shell scripting, and tool-use dialogue, using four agent models across three model families. Our contributions are:

- We introduce *budgeted active feedback acquisition over decision points* and a label-free signal, $I(A;O)$, that ranks decisions by how strongly their action choices explain downstream outcomes. It remains informative when all rollouts fail and outperforms random, entropy, and success-based selection at matched budgets (§3.1, §5.2).
- We characterize the *finite-sample identifiability* of this signal. With k repeated rollouts and a minimum of m observations per action, outcome-constant and outcome-separating decisions are exactly distinguished with any threshold $t < H_2(m/k)$, while between these extremes the score measures how much behavioral variation the outcome resolves. The same analysis proves success-based acquisition blind when all rollouts fail (§3.2, Appendix A).

- We make expert feedback *non-regressive* through selective application: an error gate withholds a learned rule from already-solved attempts and activates it only when its antipattern is present, eliminating win-to-lose flips even when always-on context costs a weak agent up to 17.6pp (§3.4).
- Across 4 benchmarks $\times$ 4 agent models with an equal expert budget, SynAct improves accuracy in all 16 settings and leads four experience-distillation baselines in 15. The same signal transfers to inference-time routing, where it beats budget-matched cascade, uncertainty, and expert-judge routers in 15 of 16 settings (§5).

2 Related Work

Self-improving agents. A large body of work builds reusable context from an agent's own traces, including playbooks (Zhang et al., 2026), reasoning and causal memories (Ouyang et al., 2026; Majumder et al., 2024; Zhao et al., 2024), induced workflows and tools (Wang et al., 2025; Kujanpää et al., 2026), and self-written guidelines and manuals (Fu et al., 2024; Chen et al., 2024b). These form our primary baselines, and they differ from SynAct on two axes. Their supervision reduces to the agent's own judgment of its traces, which is least reliable where the agent is confidently wrong (Huang et al., 2024), and several bootstrap only from successes (Wang et al., 2025; Zhao et al., 2025), leaving no signal on tasks the agent fails throughout. They also inject context unconditionally, a regime known to introduce regressions (Shao et al., 2026). Reflective prompt search (Agrawal et al., 2026) is complementary, optimizing prompt space rather than allocating supervision. SynAct instead spends a small metered expert budget and gates what it learns.

Test-time correction and search. Self-generated feedback (Shinn et al., 2023; Madaan et al., 2023), search over trajectories (Yao et al., 2023a; Zhou et al., 2024), and compute-optimal allocation (Snell et al., 2025) scale the agent's own computation, which can recover slips yet cannot supply knowledge the agent lacks (Huang et al., 2024). Relevant to our mechanism, Chen et al. (2026) distinguish syntactic misunderstanding of an environment's conventions from semantic misunderstanding of the task, and the former is what an authored convention repairs. Our premise is that one class of failures resists additional agent compute, and the contribution lies in where to spend a different resource.

Budgeted acquisition and expert intervention. We inherit from pool-based active learning (Settles, 2009), where query-by-committee measures disagreement across hypotheses (Seung et al., 1992) and information-theoretic criteria select maximally informative queries (Houlsby et al., 2011). Our k rollouts act as such a committee, but the unit of acquisition is a decision inside an episode rather than an instance, and the learner updates context rather than weights. On the intervention side, DAgger queries an expert at visited states (Ross et al., 2011), preference learning extracts training signal from scarce human judgments (Christiano et al., 2017; Liu et al., 2026b), and targeted feedback recovers full-annotation quality from a fraction of labels (Xu et al., 2025), and all three economize how much supervision is needed while leaving open where within an episode it is most informative. LEAP is the closest prior work, teaching an agent from a privileged AI expert (Choudhury & Sodhi, 2025), though it consults the teacher per episode and fine-tunes, whereas we make one expert call per iteration and leave the model frozen. Deng et al. (2026) come nearest to our score, timing a clarification request by information gain, though they query the user about intent rather than an expert about a decision.

Credit assignment and routing. Localizing the step responsible for a failure has been studied through automated failure attribution, which requires a failure label our score does not (Zhang et al., 2025), and critic-free RL methods that group repeated environment states across rollouts to sharpen step-level advantages (Feng et al., 2025; Cheng et al., 2026). The grouping instinct is shared, and the consumers differ: they convert the grouped statistics into gradient updates for an unfrozen policy, whereas we convert them into a ranking that routes a metered expert to a frozen one. Process supervision localizes credit by training a step-level verifier (Lightman et al., 2024; Wang et al., 2024), whereas we spend a few expert calls to author context, and the two could compose, with a verifier refining the outcome projection or breaking ties among equally ranked decisions. Uncertainty estimates over *answers* (Wang et al., 2023; Kadavath et al., 2022) and cost-aware model selection (Chen et al., 2024a; Ong et al., 2025; Yue et al., 2024) both operate on whole queries. We instead allocate expert attention within a query, at the decision whose alternatives separate the outcomes.

3 SYNACT: SYNERGISTIC ACTIVE FEEDBACK LEARNING

The central challenge is deciding *where expert attention is worth spending*. Given a fixed *acquisition budget* B, we **acquire** the B decision points that appear most consequential, have the expert **author** a reusable rule distilling what those decisions reveal, and **apply** that rule only when the current failure matches the learned error pattern. We make only one expert call per iteration. The objective is therefore to maximize the value obtained from each expert query rather than accuracy alone. We optionally repeat this process as the learned rule changes the agent's remaining failures (§3.5).

Problem setup. An *agent* is a small or medium open-weights model π that interacts with an environment through a ReAct-style loop (Yao et al., 2023b). An *expert* provides knowledge the agent does not have: a subject-matter expert in deployment, instantiated in our experiments by a strong model (Claude Opus (Anthropic, 2026)). The expert is available during training only, leaving the agent to operate alone at test time. A rollout is a sequence of state–action pairs $\tau = (s_0, a_0, \ldots, s_T)$, with $\mathrm{succ}(\tau) \in \{0, 1\}$ denoting its outcome under the benchmark checker.

3.1 ACQUISITION: FINDING CONSEQUENTIAL DECISIONS

The ideal acquisition score would tell us how much an expert correction at each decision would improve the agent. That quantity is counterfactual: estimating it requires intervening on the decision and observing what would have happened, often before we know the task's answer. We therefore extract a proxy for consequence from the agent's own behavior. We run the same task multiple times, find decisions where different runs make different choices *in the same state*, and ask whether those choices lead to different terminal outcomes. If all alternatives converge to the same outcome, the decision provides little evidence that changing it would matter. If different alternatives lead to different outcomes, the decision becomes a natural target for intervention. We capture this contrast with the mutual information $I(A; O)$ between the action at a decision and the rollout's terminal outcome class. Ranking by this signal concentrates useful decisions near the top, with the leading decile capturing most of the benefit (§5.3).

State-keyed grouping. For each training task, we independently sample k rollouts from π at temperature > 0 (k=8 by default; Appendix B.3) and group steps across rollouts by the environment state. Let $\phi(s)$ denote a state key: a normalized hash of the observation at s, canonicalized per benchmark by lower-casing, collapsing whitespace, and removing volatile fields such as step counters. A *decision node* v is a maximal set of (run, step) pairs sharing a state key, with $R(v)$ the runs reaching v, partitioned into *arms* by the next action. We score only arms supported by at least m=2 runs, since an arm observed once cannot provide reliable within-state evidence. We key on state rather than trajectory position because two runs may reach the same state through different action sequences while facing exactly the same decision, a correspondence position-based grouping discards. We additionally verify the key against an order-invariant alternative: the acquired set changes on 5% of tasks, and an expert judge deems the additional merges largely illegitimate (Appendix C).

Assumption 1 (Decision identifiability)**.** *The state key ϕ is* decision-sufficient*: if $\phi(s) = \phi(s')$ then the two states admit the same action set and induce the same conditional outcome distribution for each action. The outcome projection ω depends only on the terminal state. Conditioned on that state, O is therefore independent of A_v.*

Assumption 1 makes the comparison meaningful. The first clause ensures that states sharing a key represent the same decision, while the second ensures that the outcome projection measures what happens *after* the decision rather than encoding the decision itself. Without the latter, an ω that reads the branching action would make every decision consequential by construction. Deterministic environments sharpen the conditional outcome distributions but guarantee neither clause: both depend on the sufficiency of ϕ and ω (Appendix A.7).

Outcome projection. The outcome variable should capture a decision's consequence without requiring a gold answer. Let A_v be the action taken at v and $O = \omega(\tau)$ the rollout's *terminal outcome class*, computed from its own trajectory. We assign the same class to runs that end in the same state. In practice, we derive O from the normalized final `SELECT` on SQL, sorted `craft`/`get` targets on TextCraft, the final non-navigation command on Bash, and the write-tool multiset on τ-bench. Coarsening or refining these projections shifts the ranking but leaves the authored rule unchanged or improves it (Appendix D).

Acquisition score. We now measure how strongly the terminal outcome distinguishes the alternatives at a decision. Let $\mathcal{A}_m(v) = \{a : |R_a(v)| \geq m\}$ denote the supported arms and $R_m(v) = \bigcup_{a \in \mathcal{A}_m(v)} R_a(v)$ the runs they contain. Estimating both entropies by plug-in counts over $R_m(v)$ gives

$$\mathcal{I}(v) = \hat{H}(O) - \sum_{a \in \mathcal{A}_m(v)} \frac{|R_a(v)|}{|R_m(v)|} \hat{H}(O \mid A{=}a), \tag{1}$$

the standard decomposition $I(A;O) = H(O) - H(O \mid A)$ evaluated on the supported population. The restriction to supported arms keeps the arm weights normalized and ensures $\mathcal{I}(v) \geq 0$. We retain the highest-scoring node from each task, rank tasks by that score, and send the top B to the expert. Acquisition therefore requires neither a gold label nor a success check: the agent's own repeated behavior determines where expert attention is most likely to matter.

3.2 Identifiability Under Repeated Sampling

The acquisition score comes from only k rollouts. The natural question is what it can reliably identify at finite sample sizes. We begin with two extremes the score should distinguish: decisions whose alternatives converge to one outcome and decisions whose alternatives lead to disjoint outcomes. Once a node is scorable, this distinction is exact for every realization.

Definition 1 (Outcome-constant and outcome-separating nodes)**.** *A scorable node v is* outcome-constant *if ω is constant on $R_m(v)$, so its runs reach one outcome class whichever arm they take. It is* outcome-separating *if the arms of $\mathcal{A}_m(v)$ have disjoint outcome supports, that is $\omega(R_a(v)) \cap \omega(R_{a'}(v)) = \emptyset$ for $a \neq a'$, so the outcome determines which arm was taken.*

Proposition 1 (Exact separation at finite k)**.** *Fix $m \geq 2$ and let v be scorable, meaning $|\mathcal{A}_m(v)| \geq 2$. If v is outcome-constant then $\mathcal{I}(v) = 0$ for each realization of the rollouts, and if v is outcome-separating then $\mathcal{I}(v) = \hat{H}(A_v) \geq H_2(m/k)$, with H_2 the binary entropy in bits. Hence any threshold $t \in \big(0, H_2(m/k)\big)$ classifies both types with zero error for every realization, and at the default $k{=}8$, $m{=}2$ the admissible range is $t \in (0, 0.811)$.*

We prove both clauses in Appendix A.2, and we can read the result directly off Equation 1: an outcome-constant node has $\hat{H}(O) = 0$, an outcome-separating node has $\hat{H}(A_v \mid O) = 0$ so its mutual information reaches $\hat{H}(A_v)$, and the support floor supplies the finite-sample gap between them, since any arm appearing in at least m of the k rollouts forces $\hat{H}(A_v) \geq H_2(m/k)$. Sampling noise therefore cannot collapse the two extremes. What sampling does control is *scorability*: since $N_{v,a} \sim \mathrm{Binomial}(k, p_{v,a})$, we retain an arm taken half the time with probability 0.965 at $k{=}8$, but one taken a tenth of the time only with probability 0.187 (Proposition 3). The support floor thus trades coverage for reproducibility, keeping the alternatives a written rule can address.

Partial identifiability. Most decisions lie between the two extremes. In this regime, $\mathcal{I}(v) = \hat{H}(A_v) - \hat{H}(A_v \mid O)$ (Appendix A.1): the score measures the fraction of behavioral variability at v that the outcome resolves, identifying a decision to the extent that its alternatives produce distinguishable outcomes (Proposition 2, Appendix A.3). The plug-in estimator also carries a positive null bias, $\mathbb{E}[\mathcal{I}(v)] > 0$ even for outcomes independent of the action. We use $\mathcal{I}(v)$ to *rank* decisions rather than as a hypothesis test. We compute the estimator's exact null distribution, its growth with the number of outcome classes, and its ranking stability at the default budget in Appendix A.6.

Why outcomes rather than actions or success? Action entropy $\hat{H}(A_v)$ detects behavioral variation but cannot tell whether it matters: an outcome-constant and an outcome-separating node can share the same action entropy, while $\mathcal{I}(v)$ is 0 for the former and $\hat{H}(A_v)$ for the latter. Success creates the opposite problem: replacing O with $\mathrm{succ}(\tau)$ makes an all-fail task outcome-constant, hiding exactly the tasks for which the agent most needs external knowledge (Proposition 4, Appendix A.5). Our outcome projection instead preserves distinctions among *wrong* end states, keeping the signal informative even when no rollout succeeds. Empirically, $I(A;O)$ routes $1.9\times$ more decisions than a success-based signal (274 against 146 on one SQL model), with 20 of 49 acquired Qwen TextCraft tasks coming from all-fail cases (Appendix G).

3.3 AUTHORING: FROM EPISODIC EVIDENCE TO SEMANTIC RULE

Acquisition identifies *where* expert attention is valuable, and authoring determines *what* reusable knowledge that evidence supports. For each acquired node, we construct an evidence block containing the task goal and two arms with different terminal outcomes, each with its diverging action, the trajectory that followed, and a `correct`/`wrong` tag from its success flag. When no rollout succeeds, we instead mark the block `ALL-FAIL` and provide two distinct wrong end states. Acquisition thus remains label-free, while authoring may use success annotations where they exist.

A single decision can expose a mistake without revealing whether that mistake recurs across tasks. We therefore aggregate all acquired blocks and give the expert the full digest in **one call**. We ask the expert to *diagnose* recurring failure modes, *count* the blocks each mode explains, mark those a fixed convention could repair, and *write* one convention of 2–4 imperative lines covering the dominant repairable modes, with individual bullets free to address distinct modes, plus one *antipattern* summarizing the family of mistakes the convention prevents. This aggregation turns isolated episodes into a transferable rule, and the single call decouples expert cost from the acquisition budget: more evidence costs no additional expert invocations.

3.4 APPLICATION: SELECTIVE INJECTION BEHIND AN ERROR GATE

A useful convention can still hurt when applied to the wrong failure. SynAct lets the current attempt determine whether the learned rule should be used on the next one. Each task receives at most two attempts. We first run the agent with no convention at temperature 0 and stop if it succeeds. Otherwise, an LLM gate inspects the failing trajectory for the learned antipattern. Only when the gate fires do we prepend the convention and retry once. The gate sees neither the success bit nor the reference solution. We use the agent model for gating rather than the expert, so this step adds no expert cost (Appendix B.4).

This design gives a pointwise non-regression guarantee: since a base success never sees the convention, the rule cannot convert it into a failure, and any improvement must come from repairing failures. We report accuracy as base successes plus gate-repaired failures and track n_{fired}, the tasks reaching the gated retry. In Appendix J we evaluate the gate against unconditional injection, validate its firing decisions against an expert judge, redeploy the baselines' knowledge behind the same gate, and substitute a weaker authoring expert.

3.5 ITERATION AND INFERENCE-TIME REUSE

Iterative acquisition. Learning a rule changes the failure distribution. Once the first convention repairs some errors, previously hidden failure modes become more prominent, and the original rollouts cannot reveal this shift because they were generated before the rule existed. SynAct therefore re-collects after each update. At iteration 1, we acquire from base rollouts. At iteration $i > 1$, we generate k fresh rollouts with the current convention active, score the residual failures, and ask the expert to *revise* the convention: retaining bullets supported by the new evidence, adding bullets for newly surfaced modes, and removing bullets the evidence contradicts. We rewrite the antipattern alongside the convention so that the gate always evaluates the current rule, and each iteration thus spends its single expert call on what remains unrepaired. We save and evaluate every iteration's rule independently, turning iteration into a budget–performance curve rather than a fixed schedule (§5.3).

Inference-time variant. The same signal can allocate expert attention directly when the expert remains available at deployment, as in an LLM cascade (Chen et al., 2024a; Yue et al., 2024). Because $\mathcal{I}(v)$ is computed from k rollouts of the task at hand, the separation result applies under the same conditions with no training set. Instead of amortizing the feedback into a rule, SynAct spends the signal immediately. We first draw k rollouts, and if the task is *behaviorally consistent*, meaning its modal outcome class covers at least a fraction θ of runs, we return the modal answer at no further cost. Otherwise, under a budget permitting expert consultation on a fraction B of tasks, we send the top B inconsistent tasks by $\mathcal{I}(v^\star)$ to the expert, which supplies a correction at $v^\star$ for the agent to apply on retry, and the remainder receive additional self-reflection (Shinn et al., 2023) at a matched attempt budget, equivalently a per-benchmark threshold on $\mathcal{I}(v^\star)$.

Consistency thus determines *whether* to intervene, while $\mathcal{I}(v^\star)$ determines *where* expert attention is worth spending: SynAct reserves the expert for cases where behavioral disagreement points to a consequential decision, and resolves the remainder with cheaper self-reflection. Offline, the expert teaches once and the resulting rule deploys without the expert, while online the expert is consulted per difficult task and needs no training set (§5.4).

4 Experimental Setup

Benchmarks. We evaluate on four agent benchmarks with binary success checks: InterCode-SQL on Spider (n_{test}=150 (Yang et al., 2023; Yu et al., 2018)), TextCraft at recipe depth 3 (n=58 (Prasad et al., 2024)), InterCode-Bash on nl2bash (n=40 (Yang et al., 2023; Lin et al., 2018)), and τ-bench retail (n=115 (Yao et al., 2024)), training on a disjoint split of up to 50 tasks each. We select these four because each exposes an observable state that recurs across rollouts, which is what makes a state key well defined under Assumption 1, and because they span text-to-SQL, compositional crafting, shell scripting, and tool-use dialogue (Appendix B.1).

Agent models and expert. We run four models from three families, Qwen3-32B (Yang et al., 2025), Gemma-3-12B and Gemma-3-4B (Gemma Team, 2025), and Ministral-3-3B (Mistral AI, 2025), and substitute Claude Opus (Anthropic, 2026) for the subject-matter expert, held fixed across all 16 settings and iterations. The substitution is conservative: the expert sees only the acquired decision points and returns a short convention, an interface a human could equally serve, though whether one would author the same convention is untested. We span weak-to-mid agents, where expert feedback has the most headroom, and Qwen3-32B is the strong control (Appendix B.2).

Baselines. We compare against four experience-distillation baselines and give each the same expert SynAct receives. The comparison thus isolates *where* expert attention is pointed from *whether* it is available: **ACE** (Zhang et al., 2026) grows an evolving playbook of tactics prepended to the prompt, **ReasoningBank** (Ouyang et al., 2026) retrieves distilled strategies from a memory of past episodes, **AWM** (Wang et al., 2025) induces workflow snippets from successful trajectories, and **AWM-iter** runs AWM for ten iterative expert induction rounds, matching our iteration count. We tune setup parameters alone and leave the methods unmodified (Appendix B.5). In §5.2 we add *acquisition-function* baselines holding our pipeline fixed and varying only the selection rule: SynAct-Succ ($I(A; \text{succ})$), SynAct-Ent (action entropy), and SynAct-Rand (uniform).

Protocol. We report each setting as the mean of 3 repeats on the full test set, holding the rule, gate, and test set fixed except for the ablated component. The gate re-measures the unaided agent at each iteration, letting us estimate measurement noise directly: base accuracy varies by 3.4–8.5pp across iterations. We pool base where possible and treat differences of this size between adjacent iterations as indistinguishable (Appendix B.6). Agent and expert tokens are reported separately because expert tokens cost roughly 10–30$\times$ more; turns are exact and tokens estimated (Appendix N).

5 Results

5.1 Routed Expert Feedback Against Experience Distillation

Accuracy against experience distillation. Distilling experience into context is the prevailing way to improve a frozen agent. We first ask what routing the same expert to decision points buys over distilling with it. We give ACE, ReasoningBank, AWM, and ten-round AWM-iter the identical expert and hold the training split, test set, and agent fixed (Figure 2, Table 5). SynAct improves on base in all 16 settings, by $+8.3$pp on average after a single iteration and $+11.7$pp at the best iteration. Taking the strongest baseline per setting, an oracle over the four, yields $+4.4$pp on average, while no fixed baseline exceeds $+1.0$pp. SynAct at one iteration therefore nearly doubles the gain of even the per-setting oracle. Exact sign tests over the 16 settings confirm the advantage against base, each baseline, and the oracle at $p \leq 2.1 \times 10^{-3}$ (Appendix L). At its best iteration, SynAct leads all four baselines in 15 of 16 settings, the exception being Ministral TextCraft, where AWM reaches 20.7 against our 19.5. The largest gains occur where the base agent is weakest, $+24.8$pp for Gemma-4B on Bash and $+18.0$pp for Ministral on SQL, which identifies SynAct's most favorable regime: agents that can reach informative end states yet lack the knowledge to reach the correct one. The

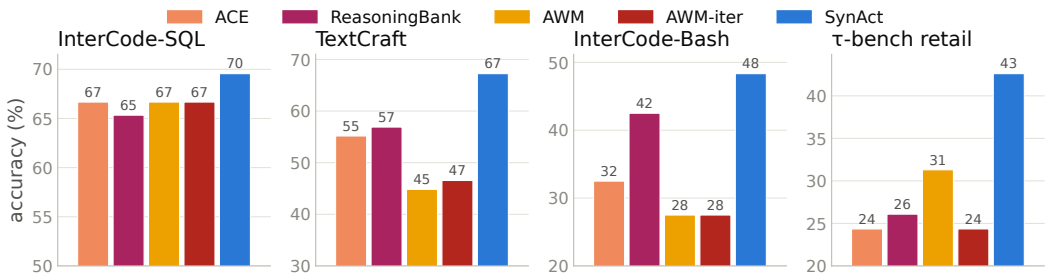


Figure 2: **Training-time methods with an equal expert budget** (agent Qwen3-32B, one panel per benchmark): ACE, ReasoningBank, AWM, AWM-iter, and SynAct at its best iteration, all given the same expert. Table 5 reports all four agent models.

controlled comparison points to *where* attention is spent: distillation summarizes whole trajectories, whereas SynAct routes the expert to the decisions determining outcomes (Appendix I).

Downside risk. Mean accuracy conceals whether a method repairs failures or exchanges them for new errors. To separate the two, we compare each method's worst setting to its base. SynAct stays at or above base within noise in every setting, whereas always-on injection can substantially reduce accuracy: ReasoningBank costs Gemma-4B 17.6pp on SQL (47.6 → 30.0), and ACE costs Ministral 3.6pp on the same benchmark. The damage concentrates on the weakest agents, the population with the most headroom, and the distinction is selective application. Holding the rule fixed and varying only its application, unconditional injection yields 32 to 36 win→lose flips on SQL, compared with 0 when gated. We therefore deploy the gate without an additional validation step, and we further validate its firing decisions against an expert judge, which the gate rarely misses (Appendix J).

5.2 Comparison of Acquisition Functions

Consulting a strong model helps on its own. We therefore ask whether the choice of decisions drives the gain. We hold the rollouts, authoring call, and consistency gate fixed, varying only the selection rule. Outcome-MI is the best rule in all 8 swept settings and all 23 pairwise comparisons, gaining +12.5pp over base, compared with +8.2 for random selection, +6.9 for success-MI, and +6.8 for entropy. The entropy comparison reveals why outcome information matters: when two actions differ frequently while converging on the same outcome, ranking by disagreement alone spends the budget on variation without consequence, precisely the distinction Proposition 1 captures. Success-MI fails in the complementary regime, remaining silent on all-fail tasks where the success label is constant (Appendix G). Under the final protocol, the four rules yield indistinguishable one-iteration gains, suggesting that outcome information matters where the theory predicts: all-fail coverage and per-task routing, while the training loop is largely insensitive to the ranking (Appendices F and P).

5.3 Expert-Token and Sampling Budgets

Cost of a routed expert budget. With the expert held fixed, we next measure how much accuracy each method buys per expert token. Figure 3 plots SynAct's gain over base across ten iterations against the cumulative expert tokens spent authoring the rule, with the one-shot distillers at their own measured budgets. The distillers cluster near zero gain at comparable spend, whereas SynAct's gain grows through the first several iterations before flattening: routed expert tokens keep buying accuracy after the distillers have saturated. Expert turns diverge further, one call per iteration against one or more per task (Appendix M).

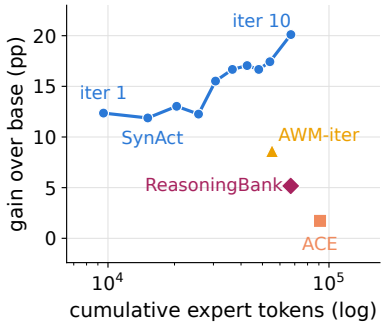


Figure 3: **Gain vs. cumulative expert tokens** (TextCraft, Qwen3-32B, log x): SynAct over ten iterations, distillers at their measured budgets.

Sensitivity to sampling budget and iteration count. Since rollouts dominate the cost, we turn both knobs down.

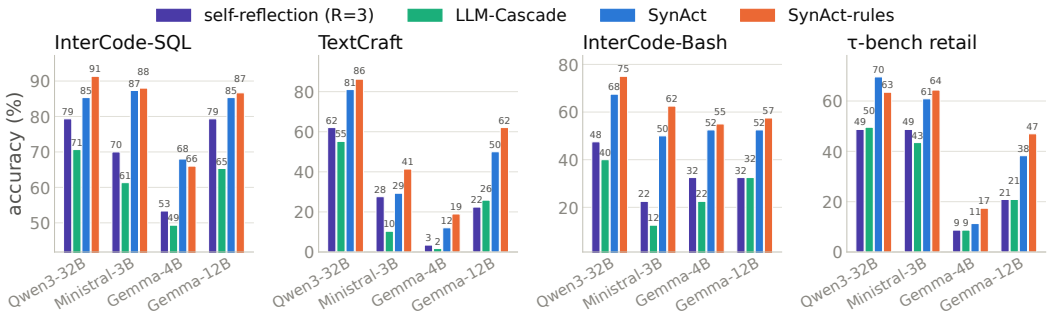


Figure 4: **Inference-time routing at a matched expert budget** (θ=0.6, R=3, one panel per benchmark): self-reflection with ten retries, a token-matched cascade, SynAct, and SynAct with the iteration-10 rule active. Table 9 adds the action-entropy and expert-judge routers.

All six swept settings beat base at k=2, a 4× cheaper collection, and the top decile of MI-ranked decisions already delivers most of the full-budget gain (Table 4, Appendices F and H). Iteration then compounds what a single pass leaves: the first iteration captures 70% of the median setting's best-iteration gain, and later passes lift Qwen TextCraft from +11.4pp to +20.6pp. Across all 160 iteration evaluations, exactly one dips below base, by 0.2pp within noise (Table 7, Appendix K).

5.4 Inference-Time Routing

We finally test whether the allocation principle transfers to inference, matching each router to SynAct's budget (§3.5) and varying only its aim: a token-matched cascade (Chen et al., 2024a; Yue et al., 2024), an action-entropy router, and an expert judge routing tasks itself (Figure 4, Table 9). SynAct or its rule-augmented variant is the best router in 15 of 16 settings, and on tasks both routers admit, the outcome-MI branch rescues up to 2.7× more (Appendix P). Against self-reflection with ten retries (Shinn et al., 2023), three attempts plus routed consultation win 14 of 16 settings and add +11.6pp on average: one aimed correction outperforms seven unaimed retries. The token-matched cascade never exceeds the retry baseline, consistent with whole-answer escalation wasting budget on localized failures.

The learned rule and online routing also compose: activating the iteration-10 convention alongside routing lifts accuracy in 14 of 16 settings over routing alone, by +5.1pp on average and +12.1pp for Gemma-12B TextCraft, as the standing rule repairs the recurring mode and frees the routed call. Raising the retry budget from R=3 to 10 never hurts and helps 11 of 16 settings (Appendix O).

6 Conclusion

We recast improving an LLM agent as active learning over decision points: under scarce expert attention, the question is not which trajectories to review, but which decisions are worth explaining. SynAct answers with the mutual information between an action at a state and the terminal outcome class, which under decision-sufficient projections exactly separates outcome-separating from outcome-constant decisions at finite k while remaining informative where success-based scores are blind. One expert call per iteration turns this signal into a gated convention improving all 16 settings, while four distillers using the same expert budget average under +1pp. The same signal also routes consultation at deployment, suggesting a natural next step: closing the loop with human experts, where turns rather than tokens are the scarce resource.

Limitations. Because Assumption 1 constrains the projections rather than the dynamics, SynAct applies to stochastic environments, though we do not measure that degradation. The label-free claim covers acquisition alone, and the consistency gate can miss confidently wrong but behaviorally consistent failures. Our expert is a proxy for a strong human, leaving the cost of human authoring untested, and the support floor can miss rare modes (Appendices J, A.4).

AI USE STATEMENT

In this work, we used generative AI tools for designing or providing feedback on research methodology or experiments, implementing methods, and interpreting results. We have not used generative AI tools for generating synthetic data sets, developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, or proposing or refining hypotheses. Assisting with translation, cleaning and reformatting datasets, and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for editing the paper to improve readability and for identifying relevant literature. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness, AI-suggested interpretations were checked against outputs, and AI-identified references were verified against their source records. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The method is specified completely in §3, covering the state key and grouping (§3.1, Appendix C), the per-benchmark outcome projections ω (Appendix D), the acquisition score with its $m \geq 2$ arm floor (Eq. 1), the verbatim authoring prompt and output schema (Appendix E), and the two-attempt gated application protocol (§3.4). Appendix B specifies splits, step limits, models, hyperparameters, the gate, baseline configurations, and the measurement protocol. All benchmarks are public (Yang et al., 2023; Prasad et al., 2024; Yao et al., 2024) and agent models are served through a public API. Each number in the iteration and budget tables comes from a script that refuses to print any entry lacking all 3 repeats: partially-evaluated results keep moving, and no number is transcribed by hand. The harness also distinguishes a task the agent failed from a call that never completed, retrying transport and throttling errors rather than recording them as failures.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2507.19457.

Anthropic. Claude Opus 4.6 System Card. `https://www.anthropic.com/claude-opus-4-6-system-card`, February 2026. Accessed 2026-08-17.

Arthur Chen, Zuxin Liu, Jianguo Zhang, Akshara Prabhakar, Zhiwei Liu, Shelby Heinecke, Silvio Savarese, Victor Zhong, and Caiming Xiong. Test-time adaptation for LLM agents via environment interaction. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2511.04847.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024a. ISSN 2835-8856. Featured Certification. arXiv:2305.05176.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. AutoManual: Constructing instruction manuals by LLM agents via interactive environmental learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024b. arXiv:2405.16247.

Xin Cheng, Shuo He, Lang Feng, Haiyang Xu, Ming Yan, Lei Feng, and Bo An. Beyond trajectory-level attribution: Graph-based credit assignment for agentic reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2605.26684.

Sanjiban Choudhury and Paloma Sodhi. Better than your teacher: LLM agents that learn from privileged AI feedback. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2410.05434.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. arXiv:1706.03741.

Mengyi Deng, Zhiwei Li, Xin Li, Tingyu Zhu, Ying Zhao, Zhijiang Guo, and Wei Wang. Uncertainty-aware clarification in LLM agents with information gain. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2606.03135.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2505.10978.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2403.08978.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.01798.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022.

Kalle Kujanpää, Ning Liu, Shahnawaz Alam, Yeshwanth Reddy Sura, Tianyu Yang, Kristina Klinkner, and Shervin Malmasi. Tool-making and self-evolving LLM agents in low-latency systems. *arXiv preprint arXiv:2607.08010*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2305.20050.

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. NL2Bash: A corpus and semantic parser for natural language interface to the Linux operating system. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC)*. European Language Resources Association (ELRA), 2018. arXiv:1802.08979.

Ning Liu, P Aditya Sreekar, Kalle Kujanpää, Zhaoxuan Zhu, Kaiwen Liu, Chuanneng Sun, Jorge Marchena Menendez, Matthew Bales, Tianyu Yang, Shahnawaz Alam, et al. Eluna: An agentic LLM system for automating warehouse operations with reasoning and task execution. *arXiv preprint arXiv:2607.08960*, 2026a.

Ning Liu, Chuanneng Sun, Kristina Klinkner, and Shervin Malmasi. Beyond pairs: Your language model is secretly optimizing a preference graph. *arXiv preprint arXiv:2605.08037*, 2026b.

Ning Liu, Lu Zhang, Tian Gao, and Yue Yu. Disentangled representation learning for parametric partial differential equations. In *International Conference on Learning Representations (ICLR)*, 2026c. arXiv:2410.02136.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2303.17651.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. CLIN: A continually learning language agent for rapid task adaptation and generalization. In *Conference on Language Modeling (COLM)*, 2024. arXiv:2310.10134.

Mistral AI. Ministral 3. `https://docs.mistral.ai/models`, December 2025. Ministral 3 3B (v25.12), Apache 2.0. Accessed 2026-09-21.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2406.18665.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2509.25140.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. ADaPT: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4226–4252, 2024. doi: 10.18653/v1/2024.findings-naacl.264.

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635, 2011. arXiv:1011.0686.

Burr Settles. Active learning literature survey. Computer Sciences Technical Report TR1648, University of Wisconsin–Madison Department of Computer Sciences, 2009.

H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory (COLT)*, pp. 287–294, 1992. doi: 10.1145/130385.130417.

Shuai Shao, Qihan Ren, Dongrui Liu, Chen Qian, Boyi Wei, Dadi Guo, Jingyi Yang, Xinhao Song, Linfeng Zhang, Weinan Zhang, and Jing Shao. Your agent may misevolve: Emergent risks in self-evolving LLM agents. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2509.26354.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2408.03314.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-Shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024. doi: 10.18653/v1/2024.acl-long.510.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2203.11171.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911, 2025. arXiv:2409.07429.

Yifei Xu, Tusher Chakraborty, Emre Kiciman, Bibek Aryal, Srinagesh Sharma, Songwu Lu, and Ranveer Chandra. RLTHF: Targeted human feedback for LLM alignment. In *International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 69096–69115, 2025. arXiv:2502.13417.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023. arXiv:2306.14898.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a. arXiv:2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b. arXiv:2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3911–3921, 2018. doi: 10.18653/v1/D18-1425.

Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. Large language model cascades with mixture of thought representations for cost-efficient reasoning. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.03094.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2510.04618.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? On automated failure attribution of LLM multi-agent systems. In *International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76583–76599, 2025. arXiv:2505.00212.

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

Wanjia Zhao, Mert Yuksekgonul, Shirley Wu, and James Zou. SiriuS: Self-improving multi-agent systems via bootstrapped reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2502.04780.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160, 2024. arXiv:2310.04406.

A IDENTIFIABILITY OF CONSEQUENTIAL DECISIONS

We characterize what $\mathcal{I}(v)$ can identify from only k rollouts, proceeding from its empirical mutual-information representation to finite-sample separation, partial identifiability, sampling recovery, ranking behavior, and the assumptions these guarantees require. For a decision node v, let $\mathcal{A}_m(v) = \{a : |R_a(v)| \geq m\}$ be the arms clearing the support floor, $R_m(v) = \bigcup_{a \in \mathcal{A}_m(v)} R_a(v)$, $N = |R_m(v)|$, and $n_a = |R_a(v)|$. We compute entropies as plug-in estimates in bits on $R_m(v)$, call v *scorable* when $|\mathcal{A}_m(v)| \geq 2$, and write $H_2(x) = -x \log_2 x - (1-x)\log_2(1-x)$. Restating Eq. 1,

$$\mathcal{I}(v) = \hat{H}(O) - \sum_{a \in \mathcal{A}_m(v)} \frac{n_a}{N} \hat{H}(O \mid A{=}a). \tag{2}$$

A.1 THE MUTUAL INFORMATION REPRESENTATION

We first identify what $\mathcal{I}(v)$ measures, and the support restriction makes the answer exact. Because both entropy terms are computed on the same population $R_m(v)$, the arm weights sum to one, and Eq. 2 is exactly mutual information under the empirical joint distribution of actions and outcomes.

Lemma 1 (Empirical joint representation)**.** *Let $\hat{P}$ be the empirical joint distribution of (A_v, O) on $R_m(v)$, that is $\hat{P}(a,o) = |\{\tau \in R_a(v) : \omega(\tau) = o\}|/N$ for $a \in \mathcal{A}_m(v)$. Then $\mathcal{I}(v) = I_{\hat{P}}(A_v; O)$, the mutual information of $\hat{P}$. Consequently*

$$\mathcal{I}(v) = \hat{H}(O) - \hat{H}(O \mid A_v) = \hat{H}(A_v) - \hat{H}(A_v \mid O), \tag{3}$$

and $0 \leq \mathcal{I}(v) \leq \min\{\hat{H}(A_v), \hat{H}(O)\}$.

Proof. The marginal of $\hat{P}$ on A_v is $\hat{P}(a) = n_a/N$, and its conditional distribution given $A_v{=}a$ is the empirical distribution of ω on $R_a(v)$. Substituting these distributions into Eq. 2 gives $\mathcal{I}(v) = I_{\hat{P}}(A_v; O)$. The identities and bounds then follow from standard properties of mutual information. Because $\hat{P}$ is fixed for each realization, no assumption about the sampling process is required. □

We can therefore reason directly about the two variables that matter: the action taken at v and the outcome reached after it. The key question is whether finite sampling gives us enough separation to identify consequential decisions.

A.2 FINITE-SAMPLE SEPARATION

We next show that the support floor creates such a separation, and it is the only place where the rollout budget k enters the bound.

Lemma 2 (Arm-entropy floor)**.** *If v is scorable then $\hat{H}(A_v) \geq H_2(m/N) \geq H_2(m/k)$, with equality in the first bound when $|\mathcal{A}_m(v)| = 2$ and the arm counts are $(m, N-m)$.*

Proof. Write $q_a = n_a/N$. Scorability gives at least two arms with $q_a \geq m/N$. Let $a^\star$ be an arm of minimum count, so $q_{a^\star} \in [m/N, 1/2]$. By the grouping property of entropy,

$$\hat{H}(A_v) = H_2(q_{a^\star}) + (1 - q_{a^\star})\, \hat{H}\left(\left. \tfrac{q_a}{1-q_{a^\star}} \right|_{a \neq a^\star} \right) \;\geq\; H_2(q_{a^\star}) \;\geq\; H_2(m/N),$$

using that H_2 is non-decreasing on $[0, 1/2]$. Equality requires the residual term to vanish with $q_{a^\star} = m/N$, that is exactly two arms with counts $(m, N-m)$.

Scorability forces $N \geq 2m$, and $N \leq k$, so $m/k \leq m/N \leq 1/2$ and monotonicity of H_2 on $[0, 1/2]$ gives $H_2(m/N) \geq H_2(m/k)$. □

We can now ask whether the score exactly separates the two decision types that matter most. It does, provided the relevant arms are observed often enough.

Proof of Proposition 1. Clause 1, outcome-constant nodes. If ω is constant on $R_m(v)$, then $\hat{H}(O) = 0$, and Lemma 1 gives $0 \le \mathcal{I}(v) \le \hat{H}(O) = 0$.

Clause 2, outcome-separating nodes. If the arms have disjoint outcome supports, then each observed outcome identifies a unique arm, so $\hat{H}(A_v \mid O) = 0$. By Eq. 3,

$$\mathcal{I}(v) = \hat{H}(A_v) - \hat{H}(A_v \mid O) = \hat{H}(A_v) \ge H_2(m/k),$$

where the last inequality is Lemma 2.

Separation. For any $t \in \big(0, H_2(m/k)\big)$, outcome-constant nodes have $\mathcal{I}(v) = 0 < t$ and outcome-separating nodes have $\mathcal{I}(v) \ge H_2(m/k) > t$, so thresholding at t separates the two classes without error. At m=2 and k=8, $H_2(2/8) = H_2(0.25) = 2 - \frac{3}{4}\log_2 3 \approx 0.8113$, giving the admissible interval $(0, 0.811)$ quoted in the proposition. □

The result requires no asymptotics. We obtain zero information when the outcome is constant, we obtain the full arm entropy when the outcome identifies the arm, and the support floor creates a finite-sample gap between the two. We also do not require each arm to produce a single outcome: an arm may produce several outcomes as long as no outcome is shared across arms. What remains is the ground between the extremes, where supports overlap.

A.3 Partial Identifiability

We rarely expect decisions to be perfectly constant or perfectly separating: arms share some outcomes while differing on others. We can characterize this intermediate regime directly through mutual information.

Proposition 2 (Component-wise partial identifiability)**.** *For a scorable node v, define the* resolved fraction $\rho(v) = \mathcal{I}(v)/\hat{H}(A_v)$ *when* $\hat{H}(A_v) > 0$. *Then*

$$\rho(v) \in [0, 1], \qquad \rho(v) = 1 - \frac{\hat{H}(A_v \mid O)}{\hat{H}(A_v)}, \tag{4}$$

so $\rho(v) = 1$ exactly when O determines the arm, recovering the outcome-separating case, and $\rho(v) = 0$ exactly when O carries no information about the arm, which includes the outcome-constant case. Furthermore

$$\hat{H}(A_v) - \mathcal{I}(v) = \hat{H}(A_v \mid O) \le \log_2\big|\mathcal{A}_m(v)\big|, \tag{5}$$

and the shortfall of $\mathcal{I}(v)$ below $\hat{H}(O)$ is the average within-arm outcome entropy, $\hat{H}(O) - \mathcal{I}(v) = \sum_a (n_a/N)\hat{H}(O \mid A{=}a)$.

Proof. By Lemma 1, $0 \le \mathcal{I}(v) \le \hat{H}(A_v)$, giving $\rho(v) \in [0, 1]$, and Eq. 4 follows by dividing $\mathcal{I}(v) = \hat{H}(A_v) - \hat{H}(A_v \mid O)$ by $\hat{H}(A_v)$. Thus $\rho(v) = 1$ iff $\hat{H}(A_v \mid O) = 0$, meaning each observed outcome co-occurs with exactly one arm, while $\rho(v) = 0$ iff A_v and O are independent under $\hat{P}$, which an outcome-constant node satisfies because O is degenerate. Equation 5 follows from Eq. 3 and $\hat{H}(A_v \mid O) \le \hat{H}(A_v) \le \log_2 |\mathcal{A}_m(v)|$, and the final claim is Eq. 2 rearranged. □

Equation 4 gives the interpretation: $\hat{H}(A_v)$ measures how much the agent's decisions vary, and $\mathcal{I}(v)$ measures the portion of that variation the observed outcome resolves. Arms reaching $\{x, y\}$ and $\{z, w\}$ at balanced counts give $\mathcal{I}(v) = \hat{H}(A_v) = 1$ bit, whereas arms reaching $\{x, y\}$ and $\{y, z\}$ share the class y and give $\mathcal{I}(v) = 0.5$. SynAct therefore spends its acquisition budget on the component of decision variation the outcome actually resolves, and the digest exposes the corresponding divergent end states to the expert.

A.4 Recovery Under Repeated Sampling

The separation result is conditional on scorability. The natural question is how often finite sampling recovers the relevant arms. A consequential arm that appears only once is invisible to the score however different its outcome, and we quantify this coverage directly.

Table 1: **Arm recovery at the default budget** k=8, m=2, from Eq. 6. An arm the agent takes in half its rollouts is retained with probability 0.965, whereas one taken a tenth of the time is retained with probability 0.187. The support floor therefore selects reproducible decision components and discards singleton behavior.

$p_{v,a}$	0.10	0.20	0.25	0.30	0.40	0.50
$\Pr[N_{v,a} \geq 2]$	0.187	0.497	0.633	0.745	0.894	0.965

Proposition 3 (Arm recovery under repeated sampling)**.** *Fix a node v and suppose each rollout is independent and reaches v and takes arm a with probability $p_{v,a}$. Then $N_{v,a} \sim \mathrm{Binomial}(k, p_{v,a})$ and*

$$\Pr[N_{v,a} \geq m] = 1 - \sum_{j=0}^{m-1} \binom{k}{j} p_{v,a}^{j} (1 - p_{v,a})^{k-j}, \tag{6}$$

which at m=2 is $1 - (1 - p_{v,a})^k - k p_{v,a}(1 - p_{v,a})^{k-1}$. If r arms each satisfy $p_{v,a} \geq p_{\min}$, a union bound gives $\Pr[\text{all } r \text{ arms clear the floor}] \geq 1 - r\big[(1 - p_{\min})^k + k p_{\min}(1 - p_{\min})^{k-1}\big]$, and v is scorable once any two arms clear it.

Proof. Each rollout contributes an independent Bernoulli($p_{v,a}$) trial, giving $N_{v,a} \sim \mathrm{Binomial}(k, p_{v,a})$ and Eq. 6. At m=2 the failure probability $(1 - p_{v,a})^k + k p_{v,a}(1 - p_{v,a})^{k-1}$ is non-increasing in $p_{v,a}$, and a union bound over the r arms at $p_{v,a} \geq p_{\min}$ gives the stated bound. Any two retained arms make v scorable. □

Table 1 evaluates Eq. 6 at the default k=8, m=2.

The support floor therefore trades coverage for reproducibility. We do not claim k=8 recovers every consequential decision: it recovers the portion of behavior supported by repeated evidence, which is also the portion a written convention can address. Raising k widens this coverage at linear rollout cost (Appendix F).

A.5 Visibility Beyond Success

We also ask whether the signal survives when success supplies no information. A natural alternative is to define O by success, and this works when some attempts succeed, with a sharp blind spot: when every attempt fails, every decision looks identical under the success projection. Outcome information carries no such limitation.

Proposition 4 (Beyond-capability visibility)**.** *Let v be scorable on a task where no rollout succeeds, so $\mathrm{succ}(\tau) = 0$ for every $\tau \in R_m(v)$. Then the success-based score is identically zero, $\mathcal{I}_{\mathrm{succ}}(v) = 0$. If in addition two arms $a, a' \in \mathcal{A}_m(v)$ have differing empirical outcome distributions, $\hat{P}(O \mid A{=}a) \neq \hat{P}(O \mid A{=}a')$, then $\mathcal{I}(v) > 0$.*

Proof. For the first claim, succ is constant on $R_m(v)$, so Clause 1 of Proposition 1 gives $\mathcal{I}_{\mathrm{succ}}(v) = 0$ for every realization. For the second, Lemma 1 identifies $\mathcal{I}(v)$ with $I_{\hat{P}}(A_v; O)$, which vanishes iff A_v and O are independent under $\hat{P}$. Independence would force $\hat{P}(O \mid A{=}a) = \hat{P}(O \mid A{=}a')$, contradicting the hypothesis, so $\mathcal{I}(v) > 0$. □

The distinction is structural rather than empirical: success-based acquisition is blind on all-fail tasks by construction, whereas outcome information stays positive whenever different arms lead to distinguishable end states. This is the mechanism behind the coverage gap measured in §5.2, including the Gemma-12B Bash setting where success-MI authors no rule at all.

A.6 Finite-Sample Ranking

The two extremes admit exact separation, and SynAct does not threshold decisions into classes. We rank them. We therefore ask what finite-sample estimation does to that ranking, and we answer under an exact null: take two arms with each run assigned uniformly, and ω drawn uniformly over

Table 2: **Exact null behavior of the plug-in score** when O is independent of A_v, so the true mutual information is 0. Computed by complete enumeration over all realizations, conditional on the node being scorable. P_{sc} is the probability of scorability. The estimator is upward-biased, and the bias grows with the number of outcome classes c, which is why we rank by $\mathcal{I}(v)$ rather than threshold it at 0.

	k			
Quantity (m=2, c=2)	4	6	8	10
P_{sc}	0.375	0.781	0.930	0.979
$\mathbb{E}[\mathcal{I}(v) \mid \text{scorable}]$	0.281	0.177	0.124	0.094
$\Pr[\mathcal{I}(v) \geq 0.3 \mid \text{scorable}]$	0.625	0.219	0.159	0.063
$\mathbb{E}[\mathcal{I}(v) \mid \text{scorable}]$, c=3	0.463	0.328	0.247	0.195
$\mathbb{E}[\mathcal{I}(v) \mid \text{scorable}]$, c=4	0.574	0.435	0.346	0.284

c classes independently of the arm, so $I(A_v; O) = 0$ for the underlying law. Enumerating all $2^k c^k$ realizations and conditioning on scorability gives Table 2.

The plug-in score is therefore not a zero-centered statistic at finite k. The bias decays roughly as $1/k$ but remains non-negligible at k=8: a node with no true dependence scores 0.124 bits on average. The bias also grows with the number of outcome classes, reaching 0.346 bits at c=4, motivating the coarse per-benchmark projections of Appendix D. Finally, the null reaches the maximum 1.0 bits with probability 0.002: no threshold strictly below the maximum eliminates false positives in this regime, and Proposition 1 is correspondingly restricted to the two extremes.

Ranking is more useful than thresholding here. Model the arm-outcome dependence as a binary symmetric channel in which the outcome class matches the arm except with flip probability ε, so the true mutual information is $1 - H_2(\varepsilon)$ bits, and compare an independent draw of a dependent node against an independent draw of a null node, each conditioned on its own scorability at k=8, m=2. The dependent node receives the strictly higher score with probability 0.995 at ε=0, 0.882 at ε=0.125, 0.692 at ε=0.25, and 0.517 at ε=0.375, where the true information is 0.046 bits and the comparison approaches chance. The ranking therefore reliably lifts strongly separating nodes above the null at the default budget and degrades gracefully toward chance as the true dependence vanishes.

These results motivate the way SynAct consumes the statistic, as an acquisition score and never as a hypothesis test: selecting the top B nodes compares scores against one another, whereas thresholding at zero would admit the entire null. The bias does vary with the support sizes, at the order of the Miller–Madow correction $(|\mathcal{A}_m(v)| - 1)(|\mathcal{O}| - 1)/(2N \ln 2)$. A general ranking-recovery guarantee would therefore need a separation condition accounting for this cross-node spread, and we do not claim one. The practical mitigation is a coarse per-benchmark ω, which holds $|\mathcal{O}|$ comparable across nodes by construction. The scorability row also explains the default budget: at k=8 roughly 93% of two-arm nodes are scorable, against 38% at k=4 (Appendix F).

A.7 Assumptions and Scope

We have so far established statistical identifiability, and neither causal effect nor repairability. The score measures association under the agent's sampling distribution: a hidden variable can induce a high score without the action being responsible, and an outcome-separating decision need not be repairable by a written convention. We use "identifiability" narrowly, for whether finite-sample rollouts separate outcome-constant from outcome-separating decisions, a weaker sense than the recovery of latent factors up to ambiguity in representation learning (Liu et al., 2026c), with the intermediate regime characterized by Proposition 2 and Appendix A.6.

Two assumptions are load-bearing, the clauses of Assumption 1. Determinism is not one of them.

State identifiability. We require ϕ to distinguish distinct situations without making recurring ones unnecessarily unique. If ϕ aliases distinct situations, as under partial observability, the arms pool incomparable decisions and $\mathcal{I}(v)$ measures a mixture. If ϕ is over-discriminative, reading timestamps or continuous values, groups become singletons and fail the support floor. We therefore need

sufficient recurrence without aliasing, and a learned representation could relax the exact-match form we use.

Outcome identifiability. We require ω to carry no encoding of the branching action: otherwise distinct arms are outcome-disjoint by construction and $\mathcal{I}(v) = \hat{H}(A_v)$, reducing the score to action entropy. We therefore restrict ω to the terminal state, ensuring that any dependence of O on A_v is mediated by where the run ended. A projection reading the actions a run performed violates this requirement, and one reading the resulting environment state satisfies it.

Determinism. We require determinism for neither condition and use it only to simplify implementation: equivalent situations produce identical observations, which lets ϕ use exact matching, and retrying without the injected rule reproduces the original failure, giving the gated protocol of §3.4 its control arm for free. Under stochastic dynamics we would instead use a resample-only control to separate repair from environmental randomness, and the stochasticity making $\mathcal{I}(v)$ informative in our experiments already comes from policy sampling at temperature 0.7 rather than from the environment.

For a new benchmark, we therefore need to redesign only ϕ and ω: ϕ must induce recurring decision contexts, and ω must preserve outcome distinctions without encoding the branching action.

B Detailed Experimental Setup

We provide the configuration needed to reproduce Table 5, complementing the summary in §4.

B.1 Benchmarks, Splits, and Step Limits

We summarize the per-benchmark configuration in Table 3. We use exact-match state keys because repeating an action sequence reproduces the observation sequence closely enough to group equivalent situations without a similarity threshold, as Assumption 1 requires. The τ-bench user simulator is the least controlled case because its replies are model-generated, and Appendix C reports the resulting grouping statistics. We fix the split seed to 42 so that train and test stay disjoint and reproducible across runs.

Table 3: Per-benchmark configuration. *Steps* is the hard cap on agent actions per rollout. *Workers* is the concurrency cap used during evaluation, which is 1 for the Docker-backed shell environment because that environment is not thread-safe.

Benchmark	n_{train}	n_{test}	Steps	Workers	Split rule
InterCode-SQL	50	150	10	8	deterministic shuffle, first 50 train, rest test
TextCraft	50	59	20	8	depth-3 recipe seeds shuffled, split in half
InterCode-Bash	20	40	10	1	`fs_1` shuffled, first 20 train, rest test
τ-bench retail	50	115	20	4	benchmark's own train/test task split

InterCode-SQL. We evaluate text-to-SQL on the Spider development set (Yu et al., 2018) through the InterCode harness (Yang et al., 2023) against a local MySQL instance. The agent issues SQL statements, observes result rows or errors, and submits a final query. We count an episode as successful when its result set matches the gold query's. The agent sees the natural-language question plus the table-to-column schema.

TextCraft. We evaluate compositional crafting on TextCraft (Prasad et al., 2024) at recipe depth 3, meaning the target item requires a three-level crafting tree. The agent issues `craft` and `get` commands and observes inventory changes. We count an episode as successful when the target item is present at the end. The prompt lists the full recipe menu and states that only raw leaf items may be obtained with `get`.

InterCode-Bash. We evaluate shell scripting on nl2bash (Lin et al., 2018) through InterCode, running each task in a Docker container. We count an episode as successful when the submitted command's stdout matches the reference command's output exactly, which makes this the most

format-sensitive of the four. Because the container environment is not thread-safe, we run all Bash evaluation single-worker.

τ-bench retail. We evaluate tool-agent-user interaction in the retail domain of τ-bench (Yao et al., 2024). The agent calls typed tools against a mock database while conversing with a simulated user, and we count an episode as successful when the final database state is correct and the domain policy wiki is respected. We run the user simulator at temperature 0 so that episodes are reproducible. This is the only benchmark of the four where an action is a structured tool call rather than a text command, and the only one requiring a native tool-calling interface, which is why we do not evaluate the two Gemma models here.

B.2 Models and Serving

We serve all agent models through Amazon Bedrock: Qwen3-32B (Yang et al., 2025), Gemma-3-12B and Gemma-3-4B (Gemma Team, 2025), and Ministral-3-3B (Mistral AI, 2025). We use Claude Opus 4.6 (Anthropic, 2026) as the expert, called with temperature 0, a 4000-token output cap, and extended reasoning enabled. We sample acquisition rollouts at temperature 0.7 and evaluate at temperature 0 (greedy), which makes the two attempts of the gated protocol differ only in whether the rule is present. We cap per-step output at 1024 tokens.

Reasoning mode. Qwen3-32B runs in thinking mode, and the trajectory text passed to the gate excludes the reasoning trace so that the gate judges actions rather than deliberation. The other three models have no separate reasoning channel.

Tool routing. All four models run on all four benchmarks, with τ-bench tool calls routed natively through each model's tool-calling interface.

B.3 Acquisition Hyperparameters

We use $k{=}8$ rollouts per training task, a minimum arm size of $m{=}2$, and acquire one decision node per task, the highest-scoring one. We vary $k \in \{2, 4, 6\}$ and the expert budget $B \in \{10\%, 25\%, 50\%\}$ of MI-ranked decisions in the corresponding ablations, holding everything else fixed. For the online variant, we use a consistency threshold $\theta{=}0.6$ to decide whether a task needs intervention at all, and we threshold $\mathcal{I}(v^\star)$ on the remainder, sweeping its operating range so that the expert-consultation rate spans the reported curve.

We cache rollouts to disk per (benchmark, model, task), letting us resume sweeps without resampling and recompute acquisition from a fixed rollout set. Each ablation re-uses the same cached rollouts wherever the setting permits, isolating the acquisition function from sampling variation.

B.4 Error Gate

Because we implement the gate as a single call to the *agent* model rather than to the expert, gating adds no expert cost, though it does add one agent call of latency to each failed attempt. We pass it the task description, the first 1500 characters of the failing trajectory, the antipattern sentence, and the rule, and it answers `FIRES` or `SKIP`. We default malformed or failed responses to `SKIP`: an infrastructure error can only withhold the rule rather than inject it spuriously. We record n_{fired}, the number of tasks reaching the gated retry, alongside each accuracy number.

B.5 Baseline Configurations

We give every baseline the same expert model SynAct uses (§4), with the expert performing the method's own distillation step. Table 5 thus compares where expert attention is allocated while holding expert strength fixed. We configure each method by its own protocol: ACE (Zhang et al., 2026) runs one epoch of reflect-and-curate over the training tasks and injects the resulting playbook always-on, ReasoningBank (Ouyang et al., 2026) distils one memory item per training task and retrieves the top $k{=}3$ at test time, AWM (Wang et al., 2025) induces workflows from successful training trajectories and injects them always-on, and AWM-iter repeats the induction for ten rounds, matching our iteration count. Consequently the expert-turn counts of Table 8 scale with the training set for the baselines and with the iteration count for SynAct. We tune only setup parameters,

namely training-set size, rollout count, epoch count, retrieval depth, and token budget, and leave each method otherwise unmodified. The self-supervised variants of Appendix I, ACE (val-select) and ReasoningBank (top-1), follow the earlier protocol of Appendix H and use no expert.

B.6 Measurement Protocol

We report each setting as the mean of 3 independent repeats over the full test set. Each repeat re-runs both attempts of the gated protocol, including the unaided first attempt, re-measuring base accuracy within every repeat and at every iteration. This lets us estimate measurement noise from the experiment itself instead of assuming it: base accuracy moves 3.4–8.5pp across iterations of the same setting. We therefore pool the base over completed iterations when reporting a base column and treat adjacent-iteration differences of that magnitude as indistinguishable at 3 repeats.

We generate per-setting numbers with a script that reads the run directory and refuses to print an entry lacking all 3 repeats: a partially-evaluated result cannot enter a table, and no number is transcribed by hand. We also record paired flip counts (win→lose and lose→win) per setting, because aggregate accuracy hides churn in which a method breaks as many tasks as it fixes.

C State-Keyed Grouping

Acquisition compares arms only when they represent the same decision, and the grouping key determines which actions SynAct treats as alternatives from a shared state. We describe the key used in our experiments and the approximation required by our cached rollout format.

The key. We define a decision node as a maximal set of (run, step) pairs sharing a state key. The intended key is a normalized hash of the observation at s, canonicalized per benchmark by lowercasing, collapsing whitespace, and dropping volatile fields such as step counters, as constrained by Assumption 1. Our cached rollouts, however, record actions rather than full observations at every step, and the implementation keys each depth by the ordered action prefix of the run. An order-invariant alternative keys on the prefix multiset $!a_0, \ldots, a_{d-1}!$, additionally merging runs that reach one state through different action orders. We audit this alternative below and find that its extra merges are usually illegitimate, precisely where they occur most often.

Why not index by position. A position index treats step d of every run as the same decision even when runs have reached different states, which happens routinely once a trajectory takes an extra exploratory action. Recovering decision context from position alone therefore requires the action history to determine the state, whereas state keying makes the context explicit. The tradeoff runs in the other direction as well: an over-discriminative ϕ fragments a node into singletons, which the support floor then discards. The key thus trades coverage against the risk of aliasing distinct decisions (Appendix A.7).

Collision audit. Does the ordered-prefix key alias distinct decisions, or split decisions it should merge? We re-group the cached $k{=}8$ training rollouts from all 16 settings under the multiset key and compare the resulting groups. Merges are structurally common in the flexible-order benchmarks: $1{,}943$ of $6{,}536$ depth groups merge more than one ordered prefix. Yet the acquired top-1 set barely changes: 28 of 565 routed tasks change (5.0%), and the per-setting overlap is at least 0.92 in 15 of 16 settings. To determine whether these extra merges are genuine, we replay the merged branches to their reached states and ask Claude Opus 4.6 whether they face the same decision. On SQL, every sampled merge is legitimate (0 of 19 judged distinct). On TextCraft, however, 90% conflate genuinely distinct decisions, different crafting sub-goals reached through different action multisets, with τ-bench between the two at 17.5%. The ordered-prefix key is therefore the correct default: where action order carries state, the order-invariant merge creates exactly the aliasing that Assumption 1 rules out.

D Outcome Projections ω

The acquisition signal needs an outcome representation that distinguishes the consequences of different actions without consulting a gold answer. We therefore project each run's terminal state into

a benchmark-specific outcome class using only its own trajectory; two runs share a class when they end in the same projected state.

- **InterCode-SQL**: the last executed `SELECT`/`WITH` statement, lower-cased with whitespace collapsed. Two runs submitting the same query share an outcome.
- **TextCraft**: the sorted set of `craft`/`get` targets the run ended with, representing what it actually made, computed order-invariantly.
- **InterCode-Bash**: the final submitted command, normalized and excluding pure navigation (`cd`, `ls`). Bash success is determined by the command's output, making the command the closest no-gold proxy for the terminal state.
- **τ-bench retail**: the multiset of write tool calls made by the run, computed order-invariantly, since τ-bench success is determined by the database end state.

How sensitive is acquisition to the choice of ω? We coarsen and refine each projection and rescore the cached rollouts. Refining is nearly rank-preserving on SQL (Spearman 0.98 to 1.0 against the default) and mildly destabilizing on TextCraft, whereas coarsening destabilizes both directions of the pipeline: on SQL, it collapses distinct queries into one class and shrinks the routed set from 43 to 18 tasks for Qwen; on TextCraft, it scrambles the ranking (Spearman 0.46 to 0.52). Rank movement does not translate into end-to-end damage, however. Authoring one iteration under the coarse projection is competitive-to-better on all four re-run settings, and on TextCraft the coarse variant lifts more than the default ($+17.2$ vs. $+13.8$pp for Qwen, $+12.1$ vs. $+6.9$pp for Gemma-12B) at matched coverage. Grouping more divergent endings per mode lets the expert surface an additional valid convention. The default projection thus sits in a usable middle, and the authored rule tolerates substantial variation in granularity.

Sensitivity to projection granularity. We further examine how the granularity of ω affects acquisition by rescoring the cached rollouts under a coarser projection (the table set touched on SQL, the count of craft actions on TextCraft) and a finer one (the raw un-normalized terminal query, the full ordered action sequence). Refining is nearly rank-preserving on SQL (Spearman $\rho \geq 0.98$ against the default) and mildly destabilizing on TextCraft (ρ of 0.62 to 0.84), while coarsening moves rankings more, shrinking the routed set on SQL from 43 to 18 tasks and dropping ρ to 0.46 on TextCraft. Yet these ranking changes do not translate into end-to-end damage: authoring and evaluating under the coarse ω matches the default on SQL and improves TextCraft for both swept models ($+17.2$ vs. $+13.8$pp for Qwen, $+12.1$ vs. $+6.9$pp for Gemma-12B at one iteration). Grouping more divergent endings into the same class gives the expert more evidence per mode. The default ω therefore lies in a forgiving region, while coarser projections remain a defensible, cheaper setting for free-form action tasks.

E Worked Example

What does the expert see after SynAct identifies a consequential decision? We trace one iteration on TextCraft with Qwen3-32B. We draw k=8 rollouts on each of the 50 training tasks, group their steps by state key, and score each node by Eq. 1. Keeping the highest-scoring node per task yields 49 scorable tasks, including 20 from tasks where no rollout succeeded and which a success-based score would have discarded (Proposition 4).

The evidence. We compress each acquired node into one digest block holding the task goal, two arms whose end states differ, the diverging action and the trajectory that followed for each, and a `correct`/`wrong` tag from that arm's success flag. Acquisition itself reads no labels, and the tags enter only here. The digest contains qualitatively different kinds of evidence.

- A task where both arms succeed by different routes, one reaching `craft 2 lime dye` through `green dye` and the other through `bone meal`. The expert sees two valid behaviors rather than a single canonical route.
- A task where both arms fail differently: one issues `craft 1 tripwire hook` before its inputs exist, and the other loops `craft 4 stick using 2 planks` six times.
- An `ALL-FAIL` task carrying two distinct wrong end states, one repeating `craft 1 granite using 1 diorite, 1 quartz` without acquiring the inputs. No arm is

labeled correct, and the divergent failures are the only evidence available. This is precisely the case in which the success bit is silent.

The rule the expert returns. From all 49 blocks in one call the expert produces the following convention, reproduced verbatim as the agent receives it.

Convention authored by the expert

```
- Before issuing any 'craft' command, verify that ALL required input
materials are already in your inventory; if not, first 'get' or craft
the missing prerequisites.
- At the start of a task, trace the full recipe tree to leaf (raw)
materials and compute the TOTAL quantity of each needed; acquire all
base materials before beginning the crafting chain.
- Never repeat the same craft or get command more than twice in a row;
if it fails or yields no progress, re-check inventory and revise your
plan.
- Work bottom-up:  craft intermediate ingredients first, then combine
them into the final product.
```

Antipattern that gates it

```
Attempting to craft items without first ensuring all input materials
are present in inventory, leading to failed crafts, wasted steps, and
infinite retry loops.
```

What the expert abstracts. The four bullets cover distinct failure modes in the acquired evidence. The first addresses crafting before acquiring ingredients, the third addresses the repeated `craft 4 stick` loop, and the second and fourth generalize the two successful routes into a plan-then-execute ordering. Crucially, the convention names none of the concrete items that appeared in training and therefore transfers to recipe trees the training tasks never contained. A per-episode repair cannot produce this abstraction, since counting modes across blocks is what identifies which failures are systematic. Prepending these four bullets behind the gate lifts accuracy in this setting by 14.2pp over base.

Application. The resulting convention is not applied blindly. At test time the task first runs without it at temperature 0, and a successful attempt terminates immediately. After a failure, the gate reads the trajectory against the antipattern, and only when it fires do we prepend the convention and retry once.

F Rollout and Expert Budget Sensitivity

SynAct spends two distinct budgets: rollouts determine how much behavioral variation we observe, and the expert budget determines how much of that variation we ask the expert to explain. We vary each independently to measure how quickly the signal saturates.

How many rollouts do we need? Equation 1 requires $m{=}2$ runs per arm before a node becomes scorable, but additional rollouts can expose arms that a smaller budget misses and make the estimate more stable. We therefore sweep $k \in \{2, 4, 6\}$ against the $k{=}8$ default (Table 4, left). All six settings already beat their base at $k{=}2$, a $4\times$ cheaper collection, and this small budget often captures most of the gain available at $k{=}6$: Ministral SQL gains $+12.0$pp at $k{=}2$ against $+14.4$pp at $k{=}6$ over a pooled base of 51.6. Accuracy rises from $k{=}2$ to $k{=}6$ in 5 of 6 settings, and the exception, Ministral TextCraft ($16.9/18.6/11.9$ on a 12.2 base), is the noisiest setting in the grid. Since k multiplies the dominant collection cost, these results favor a small rollout budget: coarse behavioral divergence carries most of the useful signal without requiring a precise probability estimate.

How much expert attention is enough? The ranking lets us truncate the expert budget without changing the acquisition rule: we expose only the top $B\%$ of MI-ranked decisions. The top decile already gains $+7.1$pp over base on average and the quarter-budget reaches $+9.2$pp, which on Qwen SQL gives 71.1 against the full-budget 68.9 of Table 5. The leading decisions therefore contain most of the dominant failure mode, making much of the remaining budget redundant.

Table 4: **The two budget knobs.** Accuracy (%, 3-repeat mean) when cutting rollout count k (k=8 is the full method) and expert budget B, the top $B\%$ of MI-ranked decisions (B=100% is full). Each panel carries its own *base*, pooled over that panel's settings because the gate re-measures the unaided agent in every run, and bold marks the best setting per row. In the mean-gain row, — marks the base columns, which have no gain by definition. All 36 results are complete at 3 repeats.

		Rollouts k				Budget B			
Bench	Model	base	2	4	6	base	10%	25%	50%
SQL	Q	65.5	69.3	69.6	**70.4**	65.9	70.9	**71.1**	66.7
	M	51.6	63.6	63.8	**66.0**	49.9	**64.4**	62.4	61.6
	12B	63.9	66.9	66.4	**68.0**	63.5	63.3	**66.4**	66.2
TC	Q	41.1	52.0	56.5	**57.1**	43.5	59.9	**61.6**	54.2
	M	12.2	16.9	**18.6**	11.9	12.2	16.4	**16.9**	14.7
	12B	16.8	28.2	24.3	**31.1**	19.0	21.5	31.1	**31.6**
mean gain over base		—	+7.6	+8.0	**+8.9**	—	+7.1	**+9.2**	+6.8

More budget does not always help. At B=50% the average gain falls to +6.8pp, below the quarter-budget on 5 of 6 settings. A larger digest contains more heterogeneous failure modes while the expert still compresses the evidence into a single convention, and beyond a point the rule becomes less focused rather than more complete. The exception, Gemma-12B TextCraft at 31.6 against 31.1, is a 0.5pp difference on a 59-task set and does not separate at 3 repeats.

The two sweeps share a pattern. We do not need exhaustive behavioral coverage to obtain useful feedback: a handful of rollouts exposes enough divergent behavior to rank decisions, and the highest-ranked decisions carry enough shared structure for the expert to form a useful convention. SynAct therefore spends compute on discovering where behavior diverges and spends expert attention on the most concentrated part of that divergence.

Where does the rollout budget actually bind? A full ten-iteration training sweep on TextCraft with Qwen sharpens the picture: k=2 degenerates completely, exposing no within-state divergence, surfacing zero branches, and authoring no rule at any iteration. In contrast, $k \in 4, 8, 16$ all learn, with iteration-10 lifts of +20.7, +27.6, and +12.1pp, respectively, within single-repeat noise. The sharp dependence is therefore the k=2 cliff, not the exact budget above it, consistent with the recovery probabilities in Proposition 3: the signal needs enough rollouts to observe self-disagreement at all, then saturates quickly. Throttling the per-iteration expert budget to the top 25% or 50% of routed tasks over the same sweep retains most of the full-budget lift (+20.7 and +17.2 vs. +27.6pp).

Sensitivity to the support floor. Rescoring the cached rollouts at m=1 and m=3 shows that the default m=2 sits at a knee. Dropping to m=1 adds acquisitions in only 4 of 16 settings, almost all on the weakest model, whereas raising to m=3 costs coverage in 10 of 16, collapsing Ministral's TextCraft set from 37 to 9 tasks. Running m=1 authoring end to end on the two most affected settings never hurts and helps TextCraft with Ministral (+5.2 vs. +1.7pp at the same firing set), because several singleton arms carry verifiably correct recipe facts that the floor would otherwise discard. The floor is thus a safe default with a real coverage cost for weak agents, matching the coverage tradeoff in Proposition 3.

Does the acquisition ranking itself carry the gain? At one iteration under the final protocol, swapping the selection rule while holding the digest format, authoring call, and gate fixed yields statistically indistinguishable full-set gains across all 16 settings (+6.6 to +7.6pp), with per-setting winners spread across all four signals. The explanation is visible in the authored rules: when one failure mode dominates a benchmark, every selector over branching points surfaces it, and on SQL all four signals converge on the same convention. The value of outcome information therefore concentrates where the theory predicts: all-fail tasks that success-based selection cannot see (§5.2) and per-task routing at inference time (Appendix P). The end-to-end training loop is nevertheless robust to the precise ranking, lowering the cost of adopting the method.

G Coverage of Acquisition Signals

How much of the gain comes from knowing *where* to ask the expert? The clean test holds the rollouts, authoring call, and gate fixed and varies only the selection rule, comparing outcome information against random selection over decision nodes, action entropy, and a success-based score (Table 6, under the single-iteration protocol of Appendix H). Outcome-MI is best in all 8 settings and all 23 pairwise comparisons, improving over base by $+12.5$pp on average against $+8.2$pp for random, $+6.9$pp for success-MI, and $+6.8$pp for entropy. Random selection is therefore a strong baseline, since a strong expert helps wherever it is pointed, and the additional margin is what pointing it at the right decisions buys. More strikingly, entropy trails random: disagreement alone spends the budget on cosmetic variation that does not separate outcomes, precisely the distinction Proposition 1 draws. Success-MI has a different failure mode: where no task contains both a winning and a losing rollout, it has nothing to acquire.

The gap lies in coverage as well as ranking quality. Accuracy in Table 6 is defined only where a signal produces evidence, and Clause 1 of Proposition 1 predicts where each signal goes silent. Success-MI requires *both* a winning and a losing rollout, and on Gemma-12B Bash zero mixed contrasts occur, no rule is authored, and accuracy stays at the 19.2 base. Outcome-MI carries no such requirement: accuracy in the same setting reaches 31.7 ($+12.5$pp), and the signal acquires $1.9\times$ more decisions overall (274 against 146 on one SQL model). This separates two properties of an acquisition signal: *quality*, how well it ranks the evidence available to it, and *coverage*, how often it produces evidence at all. Under a fixed expert budget the latter binds. A signal that goes silent on difficult tasks does not merely rank poorly: it leaves its budget unused exactly where feedback is most needed.

H Scope of the Acquisition-Function Comparison

Table 6 gives the full acquisition-function sweep on InterCode-SQL and TextCraft. We restrict the comparison to these two benchmarks because holding the rollouts, the authoring call, and the gate fixed while varying only the selection rule costs one complete training run per rule per setting. The resulting comparison is broad across models and narrower across environments: the claim that outcome information beats random, entropy, and success-based selection rests on 8 settings across two benchmarks.

This scope also explains why the base rows differ from Table 5. The sweep predates the equal-expert protocol of §5.1, leaving its absolute values incomparable across tables. Within each setting, however, the four selection rules share the same rollouts, expert, and gate, which isolates the acquisition function. The same provenance applies to the rollout and budget sensitivity grid of Table 4. The headline training results, the iteration curves of Table 7, and the routing results of Table 9 all come from the final protocol.

I Why Self-Supervised Curation Saturates

Can stronger curation close the gap without changing the source of supervision? We compare each baseline against a more selective variant under the single-iteration protocol of Appendix H: **ACE (val-select)** deploys its playbook only when it beats the empty playbook on a held-out split, and **ReasoningBank (top-1)** retrieves at most one memory. Neither changes the basic picture. ACE and ReasoningBank land within noise of base about as often as they improve on it, with a negative aggregate effect on the two weakest models on TextCraft, and the protective variants mostly trade failure for stagnation: ACE (val-select) often declines to deploy, and ReasoningBank (top-1) stays flat. The limiting factor is therefore deeper than curation, since a weak agent's traces record what it did without recording why it was wrong.

Only two settings favor a baseline, both on Gemma-3-4B: ACE on Bash (23.3 against 20.0, a margin of 1.3 tasks) and ReasoningBank (top-1) on TextCraft (4.0 against 2.8), both near the floor where under one task separates the methods. The result points to a limitation of self-supervised experience distillation: selecting better memories cannot recover information the generating agent never possessed.

J Error Gate Ablation

Gating against unconditional injection. Does the non-regression come from the authored rule or from the gate that decides when to apply it? Holding the rule fixed and changing only its application answers this directly. Under the single-iteration protocol of Appendix H, unconditional injection produces 32 to 36 win→lose flips on SQL against 0 when gated, and Table 5 shows the same pattern in aggregate, with always-on baselines regressing on several settings while SynAct regresses on none. Flip counts expose the effect more clearly than accuracy alone, since equal numbers of repairs and regressions cancel in the aggregate. The gate thus provides deployment-time protection without a held-out validation split: when the unaided attempt succeeds, the rule is withheld and cannot undo that success.

Is the served gate faithful? The gate is a single call to the agent model, and we measure how well it approximates expert judgment. On sampled failing trajectories, we compare the served gate's firing decision against an independent Claude Opus 4.6 judgment of whether the failure exhibits the antipattern. On TextCraft, Bash, and τ-bench, the gate reaches recall 1.0, firing on every failure flagged by the expert judge, at precision 0.58 to 0.86. SQL is the conservative outlier, with recall 0.22 at precision 0.50, because its failures are heterogeneous and the per-item selector often abstains. Abstention is cheap, however: force-injecting the convention into gate-declined failures repairs only 15 to 21% of them: most skipped failures would not have been fixed by the rule anyway. Firings on unrepairable failures waste a retry, 48 to 84% of firings depending on the benchmark, but because the gate is consulted only after a failure, this waste costs compute rather than accuracy: win→lose churn is 0 in every setting.

Does gating alone explain the gap? A sharper decomposition redeploys each baseline's own Opus-authored library behind SynAct's gate, changing only the deployment protocol. Gating the ACE and AWM libraries recovers roughly half of SynAct's full-set gain ($+4.6$ and $+5.0$pp vs. $+8.2$ over the 12 swept settings), showing that selective application is a substantial transferable source of benefit. The remainder comes from the knowledge itself: SynAct's rule repairs 18% of the failures it fires on, versus 11% for either gated baseline library. On InterCode-Bash, where the baseline libraries rarely trigger the gate, targeted acquisition dominates ($+11.7$ vs. $+3.1$ and $+2.5$pp). Both components matter, but their relative contribution varies by benchmark.

Does the author's strength matter? Swapping the authoring expert from Claude Opus 4.6 to Claude Sonnet 4.6, while holding the digest, gate, and evaluation fixed, lowers the mean firing-set lift from $+8.8$ to $+4.9$pp over 8 settings, with Opus ahead or tied in 7. The gap concentrates on the hardest benchmark: on Bash with Qwen, Sonnet authored a convention prescribing exactly the buggy aggregation that the evidence warned against, and the gate never matched its antipattern to a failure, and the rule silently deactivated. Author quality therefore matters, and a bad rule fails safely through inaction rather than causing damage.

K Accuracy Across Acquisition Iterations

Does more feedback keep helping? A procedure that rewrites its own context on each pass can improve in two ways: later iterations may uncover new failure modes, but they may also compound mistakes from earlier ones. We therefore run ten iterations per setting, re-rolling the training set under the active rule, and evaluate each iteration's rule independently (Table 7, Figure 5). The resulting curves are not uniformly front-loaded. The first iteration already gives $+11.4$pp on Qwen TextCraft, $+15.8$pp on Ministral SQL, and $+15.4$pp on Qwen Bash, and it captures at least 70% of the best-iteration gain in half the settings. Others continue to improve substantially: Qwen TextCraft rises from $+11.4$pp at iteration 1 to $+20.6$pp at iteration 6, and Qwen τ-bench from $+10.3$pp to $+15.2$pp. The distinction is intuitive: one convention can repair a dominant failure mode, whereas several modes require several passes. Since each iteration costs one expert call, iteration count is therefore the more attractive knob for increasing expert compute.

Does iteration introduce drift? The curves show little evidence that successive rules accumulate damage. Across the 160 iteration evaluations in Table 7, exactly one falls below its own base, Gemma-12B Bash at iteration 3 (23.3 against a 23.5 base), a 0.2pp dip within noise on a 40-task set.

The gate provides a simple explanation for this stability: when a task does not exhibit the antipattern, the rule is withheld, and an unhelpful convention cannot touch a solved attempt.

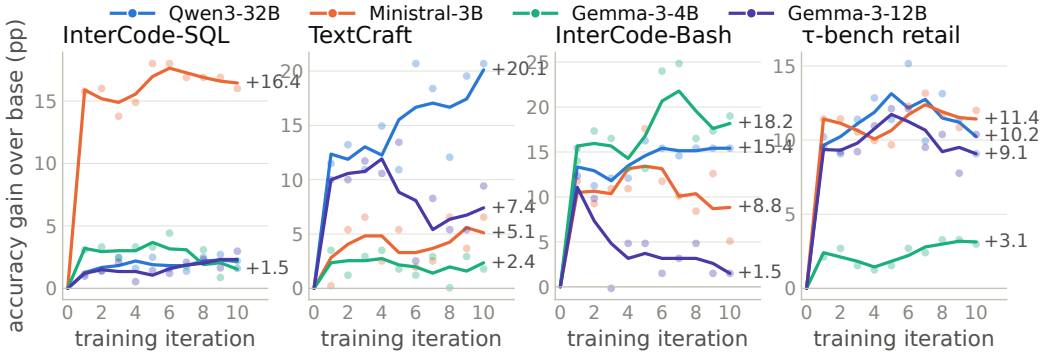


Figure 5: **Gain over the re-measured greedy base per training iteration**, one panel per benchmark and one line per agent model. Solid lines are 3-iteration moving averages and faded dots are raw 3-repetition means.

L Full Per-Iteration and Per-Acquisition Tables

Statistical reporting. The test sets are small enough that a single task visibly moves a setting's accuracy: one task changes accuracy by 0.7pp on InterCode-SQL (n=150), 1.7pp on TextCraft (n=58), 2.5pp on InterCode-Bash (n=40), and 0.9pp on τ-bench (n=115). We therefore flag near-floor comparisons where fewer than one task separates methods, treat each model–benchmark setting as the paired unit, and use exact one-sided sign tests across the 16 settings. SynAct at one iteration beats base in all 16 settings ($p = 1.5 \times 10^{-5}$), each individual baseline in 14–15 settings ($p \leq 2.1 \times 10^{-3}$), and the per-setting oracle in 14 settings ($p = 2.1 \times 10^{-3}$). Routing shows the same pattern: SynAct-rules wins all 16 settings against base and cascade ($p = 1.5 \times 10^{-5}$), 15 against SR(10) ($p = 2.6 \times 10^{-4}$), 14 against the judge ($p = 2.1 \times 10^{-3}$), and 12 against action entropy ($p = 1.8 \times 10^{-2}$). Because sign tests use only win counts, they are conservative and avoid conflating the different scales of accuracy changes across benchmarks. Within each setting, exact McNemar tests on paired per-task outcomes provide the same picture. Across 63 single-realization comparisons of SynAct's best iteration against base and the three baselines, 31 are significant at $p < 0.05$ and 37 have bootstrap 95% confidence intervals strictly above zero. Mean advantages range from +9.8 to +12.6pp, with only three negative point estimates, none significant. The largest gains are likewise robust: Bash with Gemma-4B reaches +35.0pp against ACE, with 14 discordant wins versus 0 losses, while the strongest-base settings appropriately straddle zero.

This appendix provides the two full grids summarized in the body: the acquisition-function comparison of §5.2 (Table 6) and the per-iteration curve of Appendix K (Table 7).

What makes the acquisition signal useful? Table 6 holds everything but the selection rule fixed, isolating the ranking from the rest of the pipeline. Outcome-MI wins every column, and the alternatives are informative in different ways. Random selection reaches +8.2pp over base because the expert is useful wherever it is pointed: a randomly chosen decision still lands in a real failing trajectory, and the digest still describes something worth fixing. Action entropy reaches only +6.8pp despite deliberately preferring variable decisions, and Proposition 1 explains the gap: entropy cannot distinguish a cosmetic decision from an outcome-separating one when their arm counts agree. Part of the budget therefore buys blocks whose divergent arms reach the same end state, and the expert must then explain variation that the outcome does not resolve. Ranking on the wrong quantity can be worse than not ranking at all.

Success-MI reaches +6.9pp and exposes a different limitation. It fails outright on Gemma-3-4B TextCraft, where *n/a* means no task contains both a winning and a losing rollout: under ω=succ those nodes are outcome-constant, the score is identically zero by Clause 1, and the budget goes un-

Table 5: **Training-time methods with an equal expert**, test accuracy (%, 3-repeat mean), referenced from §5.1. The shaded rows are ours. *base* for SynAct is pooled over its complete iterations, and each baseline reports its own paired greedy base. SynAct appears after one training iteration and at its best iteration, with the best iteration in parentheses. Bold marks the best learning method per column and underline the second best.

Method	Qwen3-32B	Ministral-3B	Gemma-4B	Gemma-12B
InterCode-SQL (n=150)				
base	66.8	50.9	47.6	64.6
ACE	66.7	47.3	48.0	64.7
ReasoningBank	65.3	50.7	30.0	<u>67.3</u>
AWM	66.7	56.7	34.0	65.3
AWM-iter	66.7	56.7	38.0	64.0
SynAct (1 iter)	<u>68.0</u>	<u>66.7</u>	<u>50.7</u>	65.6
SynAct (best iter)	**69.6** (9)	**68.9** (5)	**52.0** (6)	**67.6** (10)
TextCraft (n=58)				
base	46.6	13.0	2.8	19.3
ACE	55.2	15.5	<u>3.4</u>	22.4
ReasoningBank	56.9	15.5	<u>3.4</u>	15.5
AWM	44.8	**20.7**	<u>3.4</u>	17.2
AWM-iter	46.6	15.5	<u>3.4</u>	15.5
SynAct (1 iter)	<u>58.0</u>	13.2	**6.3**	<u>29.3</u>
SynAct (best iter)	**67.2** (6)	<u>19.5</u> (3)	**6.3** (1)	**32.8** (5)
InterCode-Bash (n=40)				
base	32.1	8.2	15.2	23.5
ACE	32.5	15.0	12.5	<u>22.5</u>
ReasoningBank	42.5	17.5	15.0	20.0
AWM	27.5	10.0	12.5	20.0
AWM-iter	27.5	12.5	25.0	<u>22.5</u>
SynAct (1 iter)	<u>47.5</u>	<u>20.0</u>	<u>29.2</u>	**35.8**
SynAct (best iter)	**48.3** (5)	**25.8** (5)	**40.0** (7)	**35.8** (1)
τ-*bench retail* (n=115)				
base	27.4	25.7	5.4	13.1
ACE	24.3	27.0	2.6	19.1
ReasoningBank	26.1	20.9	6.1	19.1
AWM	31.3	24.3	5.4	13.0
AWM-iter	24.3	21.7	5.2	15.7
SynAct (1 iter)	<u>37.7</u>	<u>37.1</u>	<u>7.5</u>	<u>22.6</u>
SynAct (best iter)	**42.6** (6)	**38.8** (7)	**8.7** (8)	**25.2** (6)

spent. Outcome-MI retains information whenever different actions lead to distinguishable outcomes, including when every rollout fails.

What do successive iterations learn? Table 7 evaluates each iteration's rule independently rather than committing to a schedule in advance, which lets a row be read as a curve. Two regimes recur across the sixteen rows. When one convention captures the dominant failure mode, performance peaks early and plateaus, as on Qwen Bash, which gains $+15.4$pp at iteration 1 and holds within a point of that level. When training tasks contain several distinct modes, later iterations continue to add value: Qwen TextCraft climbs from 58.0 at iteration 1 to 67.2 at iteration 6. Only one row peaks at iteration 10: a fixed schedule would either stop before the peak or spend expert calls after it, which motivates evaluating each iteration's rule separately. The *base* column moves across iterations because the gate re-measures the unaided agent each time, and we pool those measurements and treat differences of a few points as inseparable at 3 repeats.

M Cost of a Routed Expert Budget

Agent tokens against expert tokens. What does the budget look like when agent and expert computation are priced separately? SynAct spends *more total* tokens, running the agent k=8 times per training task, and roughly $15\times$ *fewer expert* tokens, 0.13% of its own budget, calling the expert

Table 6: **Acquisition-function comparison** (accuracy %, 3-repeat mean), referenced from §5.2. Everything but the selection rule is held fixed, using the same rollouts, the same authoring call, and the same gate. *n/a* means the signal acquired too little usable evidence to author a rule at all, a failure mode unique to the success-based score, which requires a task to contain both a winning and a losing rollout. The shaded row is ours, and bold marks the best rule per column.

Benchmark	Acquisition	Qwen3-32B	Ministral-3-3B	Gemma-3-4B	Gemma-3-12B
SQL	base	64.4	49.1	45.8	64.9
	SynAct-Rand (random)	69.3	64.9	50.0	67.3
	SynAct-Ent ($H(A)$)	69.3	62.2	51.3	65.6
	SynAct-Succ ($I(A;\text{succ})$)	68.2	61.3	49.6	65.3
	SynAct ($I(A;O)$)	**75.6**	**65.1**	**52.9**	**71.3**
TextCraft	base	41.2	13.0	1.7	17.5
	SynAct-Rand (random)	58.8	20.9	3.3	28.8
	SynAct-Ent ($H(A)$)	57.6	16.9	4.5	24.3
	SynAct-Succ ($I(A;\text{succ})$)	50.8	20.9	*n/a*	28.2
	SynAct ($I(A;O)$)	**67.2**	**27.1**	**6.2**	**32.2**

Table 7: **SynAct test accuracy (%) at each training iteration** (3-repeat mean), grouped by benchmark with one row per agent model, referenced from §5.3 and Appendix K. The *base* column pools the unaided measurement over the completed iterations, because the gate re-measures the base each iteration.

Benchmark	Model	base	1	2	3	4	5	6	7	8	9	10
InterCode-SQL	Qwen3-32B	66.8	68.0	68.2	69.3	68.4	69.3	68.4	68.2	69.3	69.6	68.4
	Ministral-3-3B	50.9	66.7	66.9	64.7	65.8	68.9	68.9	67.8	67.8	67.8	66.9
	Gemma-3-4B	47.6	50.7	50.9	50.0	50.9	50.9	52.0	49.3	50.7	48.4	49.8
	Gemma-3-12B	64.6	65.6	66.0	66.7	65.1	66.0	65.8	66.7	66.9	66.2	67.6
TextCraft	Qwen3-32B	46.6	58.0	59.8	57.5	61.5	57.5	67.2	64.9	58.6	66.1	67.2
	Ministral-3-3B	13.0	13.2	18.4	19.5	15.5	18.4	14.9	15.5	19.5	16.7	19.5
	Gemma-3-4B	2.8	6.3	4.0	5.7	6.3	4.6	4.0	5.7	2.9	5.7	4.6
	Gemma-3-12B	19.3	29.3	29.3	31.0	29.9	32.8	21.8	27.6	24.7	24.7	28.7
InterCode-Bash	Qwen3-32B	32.1	47.5	43.3	44.2	44.2	48.3	47.5	46.7	47.5	47.5	47.5
	Ministral-3-3B	8.2	20.0	17.5	19.2	19.2	25.8	20.0	18.3	16.7	20.8	13.3
	Gemma-3-4B	15.2	29.2	32.5	31.7	28.3	28.3	39.2	40.0	31.7	32.5	34.2
	Gemma-3-12B	23.5	35.8	33.3	23.3	28.3	28.3	25.0	26.7	28.3	25.0	25.0
τ-bench retail	Qwen3-32B	27.4	37.7	36.5	38.8	40.3	38.8	42.6	37.4	40.6	38.8	36.5
	Ministral-3-3B	25.7	37.1	37.1	36.2	35.7	35.4	38.0	38.8	37.4	36.5	37.7
	Gemma-3-4B	5.4	7.5	8.1	7.0	6.7	7.0	8.1	7.8	8.7	8.7	8.4
	Gemma-3-12B	13.1	22.6	22.3	22.3	24.1	25.2	25.2	22.6	23.5	20.9	23.5

once per iteration rather than once or twice per training task. In turns the separation widens to one expert turn per iteration against 50 to 100 per run (Appendix N). The two resources move in opposite directions, and total token count obscures the relevant trade.

Substituting a human expert. The asymmetry matters most when the expert is a person rather than a model (§4). Human throughput is bound by the interventions that can be read and authored rather than by token price. A method whose expert cost scales with the training set therefore becomes increasingly expensive to operate, whereas one intervention per iteration stays feasible under either instantiation. We measure the per-call load directly: the digest for the worked example of Appendix E spans 49 blocks and roughly 27k characters, a reading task of minutes rather than hours, and the returned convention is 2–4 lines. Expert turns are consequently the more meaningful unit for human deployment, which is why we report them alongside tokens.

Table 8: **Training cost per (model × dataset) run** at n_{train}=50 and k=8. Expert tokens are the expensive ones, costing 10–30× the agent's per-token rate. Token counts are estimates and turn counts are exact.

	Tokens		Expert
Setting	Agent	Expert	turns
SynAct, 1 iter.	7.6M	**0.01M**	**1**
SynAct, 3 iter.	22.9M	**0.03M**	3
SynAct, 10 iter.	76.2M	**0.10M**	10
ACE (1 epoch)	0.95M	0.20M	100
ReasoningBank	0.95M	0.15M	50

N Cost Accounting Details

Table 8 separates agent and expert tokens because they price differently. At one iteration SynAct spends 0.01M expert tokens against ReasoningBank's 0.15M and ACE's 0.20M, a 15 to 20× reduction. The ratio narrows to 5 to 6.7× at three iterations, since each additional SynAct iteration adds one expert call while the baselines pay their expert cost once. The largest savings therefore occur in the same regime where most of the accuracy gain is obtained, since iteration 1 captures most of the available improvement (Appendix K), and the headline 15× should consequently be read as a single-iteration figure rather than an iteration-independent property.

The token ratios also hide a second asymmetry. Expert tokens account for 0.13% of SynAct's own budget at every iteration count, against 17% for ACE and 14% for ReasoningBank: a rise in the expert's price per token barely moves SynAct's total and moves the baselines' substantially. Expert turns diverge even more sharply, one call per iteration against 50 to 100 per baseline run, and for a human expert the turn count is the binding resource. This is why we report both units.

The remaining cost is agent-side. SynAct spends 8× more agent tokens at one iteration because it runs k=8 rollouts per training task to obtain the acquisition scores. This cost is reducible: Appendix F shows k can be cut to 2 while still beating base on all six settings, whereas the baselines' expert cost is not reducible without changing the methods. Turn counts are exact, taken from cached traces, and token counts are estimates formed from exact call counts and per-call anchors measured live.

O Inference-Time Routing Details

Where should the expert intervene? At inference time the question shifts: instead of deciding which training decisions deserve explanation, we decide which test tasks deserve intervention. For each task we draw k=8 rollouts at temperature 0.7 and group them by state key as in §3.1. A consistency gate decides whether intervention is warranted: we extract an outcome key per rollout, take the modal key, and route only when the modal fraction falls below θ=0.6. A task that agrees with itself is answered by majority vote at no cost, and a task already solved by any pooled rollout is kept as is. The expert thus sees only tasks where the agent both disagrees with itself and has not already succeeded. On a routed task, $\mathcal{I}(v^\star)$ selects the decision node, the expert returns a correction anchored at that node, and the agent retries with the correction as the first of R attempts. Attempts $2, \ldots, R$ are ordinary self-reflection.

Can other routers spend the same budget as effectively? We cap every expert-using alternative in Table 9 to SynAct's expert-call count B at this operating point, making the comparison like for like. The cascade escalates whole tasks to the expert under the same expert-token spend. The action-entropy router keeps the consistency gate but replaces $\mathcal{I}(v^\star)$ with $\arg\max_v \delta H(v)$, spending its B expert plans on the most uncertain tasks. The judge hands the routing decision to the expert itself, capped at B judge calls and averaged over task-arrival orders, since the judge cannot rank tasks it has not yet seen. SynAct-rules additionally activates the iteration-10 convention from §5.3 during rollouts and retries.

Table 9: **Inference-time routing at a matched expert budget** (θ=0.6, R=3), held-out accuracy (%), referenced from §5.4. SR(10) is self-reflection with ten retries and no expert. Cascade, action-entropy, and the expert judge are each throttled to SynAct's expert budget at this operating point. Bold marks the best routing method per row.

Model	base	SR(10)	Cascade	Act.-Ent.	Judge	SynAct	+rules
InterCode-SQL							
Qwen3-32B	66.0	82.7	70.7	86.0	84.7	85.3	**91.3**
Ministral-3-3B	47.3	82.0	61.3	**88.7**	85.9	87.3	88.0
Gemma-3-4B	48.7	62.0	49.3	67.3	66.3	**68.0**	66.0
Gemma-3-12B	64.7	83.3	65.3	86.0	84.3	85.3	**86.7**
TextCraft							
Qwen3-32B	50.0	72.4	55.2	75.9	80.0	81.0	**86.2**
Ministral-3-3B	19.0	31.0	10.3	31.0	29.4	29.3	**41.4**
Gemma-3-4B	3.4	5.2	1.7	17.2	10.3	12.1	**19.0**
Gemma-3-12B	15.5	32.8	25.9	51.7	45.9	50.0	**62.1**
InterCode-Bash							
Qwen3-32B	37.5	60.0	40.0	65.0	64.4	67.5	**75.0**
Ministral-3-3B	10.0	35.0	12.5	45.0	46.1	50.0	**62.5**
Gemma-3-4B	10.0	45.0	22.5	**55.0**	51.9	52.5	**55.0**
Gemma-3-12B	27.5	47.5	32.5	52.5	55.1	52.5	**57.5**
τ-bench retail							
Qwen3-32B	25.2	65.2	49.6	67.8	67.9	**69.6**	63.5
Ministral-3-3B	27.8	62.6	43.5	58.3	61.9	60.9	**64.3**
Gemma-3-4B	5.2	9.6	8.7	13.0	11.3	11.3	**17.4**
Gemma-3-12B	12.2	29.6	20.9	33.9	37.6	38.3	**47.0**

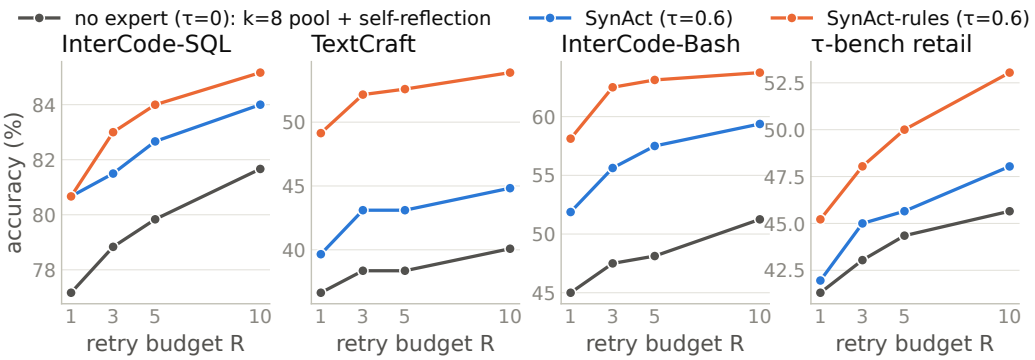


Figure 6: **Accuracy against retry budget** R, mean over the four agent models. The no-expert pool with self-reflection against SynAct and SynAct-rules at θ=0.6.

How sensitive is routing to the consistency threshold? Sweeping θ from 0.5 to 0.8 at R=3 expands the routed pool at every step, since k=8 restricts the modal fraction to multiples of 1/8. Across the two SQL settings, accuracy changes by at most 1.3pp while expert calls roughly triple, leaving little for the threshold to decide once the pool is near saturation. TextCraft with Qwen is the one swept setting where the knob matters: accuracy rises +6.9pp from θ=0.5 to 0.7 before the routable pool saturates. TextCraft with Ministral is flat because every threshold selects the same capability-bound pool. We therefore use θ=0.6 for every benchmark and model without per-setting tuning: where routing helps, a broad range of thresholds captures the gain.

What if the agent gets more chances after intervention? Table 10 raises the retry budget from R=3 to R=10 at the same expert budget and operating point, matching the retry allowance of SR(10). Accuracy never falls and rises in 11 of 16 settings, with the largest gain +11.3pp for Ministral on τ-bench. The correction and self-reflection therefore compose rather than compete: the expert supplies a targeted change, and additional attempts continue the search from there. Figure 6

makes the comparison across the full retry range, and the routed variants dominate the no-expert pool at matched R throughout: the benefit of expert feedback does not vanish once the agent is allowed more retries.

Table 10: **SynAct at R=10 against R=3**, same expert budget and operating point (θ=0.6), accuracy (%).

SQL	R=3	R=10	TextCraft	R=3	R=10
Qwen3-32B	85.3	86.0	Qwen3-32B	81.0	82.8
Ministral-3-3B	87.3	87.3	Ministral-3-3B	29.3	31.0
Gemma-3-4B	68.0	74.0	Gemma-3-4B	12.1	12.1
Gemma-3-12B	85.3	88.7	Gemma-3-12B	50.0	53.4
Bash	R=3	R=10	τ-bench	R=3	R=10
Qwen3-32B	67.5	67.5	Qwen3-32B	69.6	70.4
Ministral-3-3B	50.0	57.5	Ministral-3-3B	60.9	72.2
Gemma-3-4B	52.5	57.5	Gemma-3-4B	11.3	11.3
Gemma-3-12B	52.5	55.0	Gemma-3-12B	38.3	38.3

P Routing Signal at Inference: Outcome-MI Against Action Entropy

The training loop is robust to the acquisition ranking (Appendix F). We therefore ask where the choice of signal does matter. Inference-time routing is the natural place to look: the signal is spent per task rather than amortized into a reusable rule, and a misplaced branch point cannot be recovered through aggregation across tasks. We compare SynAct's outcome-MI branch point with the action-entropy alternative ($\arg\max_v \delta H(v)$) on τ-bench retail from two views. First, we run both routers on the full benchmark at the shared operating point (θ=0.6, R=3). Full-benchmark accuracy mixes the routing decision with the consistency gate and the unrouted pool. The second view therefore isolates the signal itself: both routers admit exactly the same B tasks through the consistency gate, differing only in *where* in the trajectory the expert plan is requested. We report the resulting rescue rate, solved over routed, on this shared task set. Gemma-3-4B is excluded because its near-floor accuracy leaves too few rescues for a meaningful comparison.

Table 11: **Outcome-MI against action entropy as the routing signal** (τ-bench retail, θ=0.6, R=3, n=115). Left: full-benchmark accuracy (%). Right: expert rescues among the B tasks both signals route, solved/routed (%). Both signals route the same B tasks per model, and the rescue columns isolate *where* the expert plan is anchored. Bold marks the better signal per view.

	Full benchmark (acc. %)			Rescues on routed set	
Model	MI-out	Act.-Ent.	B	MI-out	Act.-Ent.
Qwen3-32B	**69.6**	67.8	33	**7/33** (21.2)	5/33 (15.2)
Ministral-3-3B	**60.9**	58.3	44	**9/44** (20.5)	6/44 (13.6)
Gemma-3-12B	**38.3**	33.9	61	**8/61** (13.1)	3/61 (4.9)

On the full benchmark, outcome-MI leads action entropy by 1.8–4.4pp across the three models, although this gap reflects both the routing signal and the rest of the routing pipeline. The controlled comparison is more revealing: on the shared routed set, where the only difference is the branch presented to the expert, outcome-MI rescues 13–21% of routed tasks, compared with 5–15% for action entropy, a 1.4×–2.7× advantage that widens on the weakest model. This is the regime captured by Proposition 1: entropy identifies the step where the agent is most uncertain, which need not be the step that determines the outcome. Anchoring the expert's single correction at a cosmetic branch therefore spends the consultation without changing where the run ends.

The contrast with training-time robustness is instructive. Aggregating evidence into a digest lets the expert recover from imperfect selection, whereas a per-task consultation has no such slack. The value of a consequence-aware signal therefore concentrates where feedback cannot be amortized: at inference, choosing *where* to intervene is itself the bottleneck.