FORK OF THOUGHT: EXPLORING AND LEARNING SELF-TRIGGERED RETHINKING IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-correction without external oracle feedback has often proved unreliable for large language models in single-turn reasoning, where critiquing an output from the same conditioned context may provide little new discriminative signal. In this work, we revisit this limitation for *long-horizon agents*. Our central hypothesis is that conditioning on the full rollout history can reinforce earlier mistaken assumptions, making selective reconsideration of accumulated evidence essential to effective self-correction. We therefore formulate self-correction as an explicit tool, RETHINK, through which the main agent delegates targeted questions over a truncated history window to a helper operating in a *fresh context*, with read-only access to the environment for verification. Evaluations on Apex Agents and τ^3 benchmarks show that this intervention can improve task success across strong open-weight models, although the gains vary across domains and configurations. Controlled sampling analyses further reveal that the benefits concentrate on hard tasks and depend critically on *when* the agent invokes RETHINK and *what* context it exposes; indiscriminate calls on easy tasks can degrade performance. To learn this selective behavior, we introduce a multi-level agentic reinforcement learning framework based on *counterfactual forks*. By branching rollouts at the rethink point into non-rethink control paths, alternative advice generations, and subsequent continuations, the framework converts terminal rewards into separate relative advantages for the invocation gate, helper advice, and continuation. Experiments show that this fine-grained credit assignment helps models acquire selective, context-aware self-correction. Together, these findings identify self-correction as a learnable, state-dependent capability whose effectiveness depends on invocation timing, context selection, and the use of environment-grounded evidence.

1 INTRODUCTION

The ability to detect and repair one's own mistakes is a natural avenue for scaling test-time computation. Sampling multiple reasoning paths can improve an answer through self-consistency (Wang et al., 2022), but such gains do not establish that a model can identify and repair a mistake in its own initial solution. Extensive prior work has examined self-correction without external information, reaching a largely cautionary consensus. In single-turn question answering and reasoning tasks, prompting a model to critique and revise its own initial output yields minimal benefit and frequently degrades performance (Huang et al., 2023; Stechly et al., 2025; Kamoi et al., 2024). When a model lacks the knowledge or evidence to solve a problem initially, sampling a critique from the same context provides little new discriminative signal, and the revision step often reinforces the original mistake. Correction can instead benefit from training a dedicated corrector (Welleck et al., 2022), grounding feedback in tool execution (Gou et al., 2024; Chen et al., 2023), or using learned process verifiers (Lightman et al., 2023). These findings motivate a shift in focus from merely allocating more computation to reconsideration toward understanding what evidence supports a useful correction and how the model makes use of it.

Long-horizon agents provide a natural setting in which to investigate this question, because external feedback is already part of the task-solving process. Benchmarks such as AgentBench and WebArena expose agents to extended sequences of decisions in interactive environments (Liu et al., 2024; Zhou et al., 2023). API responses and execution results provide observations that can reveal

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.

Table 1: Prompt-based RETHINK on Apex Agents and τ^3: pass@1 (%).

Model	Apex Agents				τ^3
	Investment Banking	Law	Management Consulting	Overall	Overall
DeepSeek-V4.1-Flash	31.82	28.75	36.13	32.21	66.40
+ RETHINK	**34.85**	**30.63**	**38.06**	**34.45** (+6.94%)	**68.53** (+3.21%)
Qwen3.6-Flash	7.48	12.90	**19.23**	13.52	67.31
+ RETHINK	**12.15**	**14.19**	17.69	**14.80** (+9.43%)	**70.43** (+4.64%)
Kimi K2.5	8.33	14.38	16.25	13.27	64.53
+ RETHINK	**11.36**	14.38	**18.75**	**15.04** (+13.33%)	**66.67** (+3.31%)

discrepancies between an agent's assumptions and the environment, enabling it to adjust its actions within the same rollout (Yao et al., 2023). However, correcting an immediate action does not necessarily lead the agent to reconsider the earlier decision that caused the problem. For example, an agent may repair a malformed API call while continuing to pursue an incorrect intermediate goal; a successfully executed action likewise need not advance the user's objective. Over a long trajectory, effective recovery may therefore require connecting recent feedback to earlier assumptions, rather than only addressing the latest execution error. We hypothesize that ordinary continuation can leave such assumptions insufficiently examined, as subsequent reasoning proceeds from a history containing both environmental evidence and the agent's own prior interpretations. This motivates our central question: can an agent recover more effectively from earlier mistakes by delegating targeted retrospective reasoning to a fresh context, where a helper can selectively revisit prior interactions and verify relevant information through read-only access to the environment, rather than simply continuing from the accumulated trajectory?

Learning when to invoke an API is an established capability in language-model tool use (Schick et al., 2023). To break this conditioning, we package self-correction as an explicit, callable tool: RETHINK.[1] When invoked by the main agent with a targeted question and a specified history window, RETHINK spawns a helper instance (using the same underlying model) that reasons in a *fresh context window*. By exposing only a truncated, adjusted segment of the trajectory alongside read-only tools for environment verification, RETHINK shifts the task from full-context resampling to *de-conditioned re-reasoning*.

We evaluate prompt-based RETHINK on Apex Agents (Vidgen et al., 2026) and τ^3 (Sierra Research, 2026; Yao et al., 2024; Barres et al., 2025; Shi et al., 2026), using the same model as the main agent and as a helper in a fresh context. Table 1 reports Apex results by domain and the completed τ^3 evaluation over 375 tasks, using each model's best observed RETHINK configuration on each benchmark. On Apex, DeepSeek-V4.1-Flash (DeepSeek-AI, 2026), Qwen3.6-Flash (Alibaba Cloud, 2026), and Kimi K2.5 (Kimi Team, 2026) achieve overall relative improvements of 6.94%, 9.43%, and 13.33%, respectively. The corresponding relative improvements on τ^3 are 3.21%, 4.64%, and 3.31%. Thus, the reported overall scores increase for all three models on both benchmarks, while the Apex domain results show that the effect is not uniform across tasks. This variation motivates learning a selective invocation policy.

Despite these encouraging results, prompting alone provides limited control over how agents use RETHINK. In our experiments, agents often request broad reviews of their accumulated trajectories, with little attention to the particular decision or evidence that warrants reconsideration. Such requests may carry earlier assumptions into the helper's context and make it difficult to focus on the source of an error. This observation motivates a closer examination of *when* agents should seek advice, *how* they should formulate the question, and *how* the resulting advice affects subsequent actions. Learning these behaviors from task outcomes presents a credit-assignment challenge. A successful trajectory may reflect a well-timed consultation, useful advice, or effective execution af-

[1] Code and related data are available at `https://anonymous.4open.science/r/Fork-of-Thought/`.

ter receiving that advice. A shared trajectory-level advantage, as used in standard GRPO, provides limited resolution for distinguishing these contributions.

To investigate whether smaller models can acquire effective consultation behavior, we develop an agentic RL procedure based on *counterfactual forks*. At a RETHINK call, we sample multiple helper responses and multiple continuations under each response, together with control branches that continue without consultation. The terminal task rewards support comparisons between direct execution and rethinking, among alternative responses, and across their continuations. We use these comparisons to construct separate learning signals for the invocation, advice generation, and subsequent actions, while retaining task-level supervision for the main agent (Section 3.3). This procedure provides a way to optimize the interacting components of rethinking and examine their downstream value.

Our work investigates the conditions under which self-correction can benefit agents that accumulate evidence through interaction with an environment. Our contributions are:

- We evaluate fresh-context rethinking in long-horizon agentic settings and provide evidence of improvements in task success across the evaluated strong models. These findings extend the study of self-correction to interactive settings, where execution feedback can inform the reconsideration of earlier decisions.
- We develop an agentic RL procedure with credit assignment across invocation, advice generation, and continuation. Controlled rollout branching supports joint optimization of the main and helper roles, allowing us to study how smaller models learn to request useful advice and incorporate it into subsequent actions.
- We investigate the conditions that shape the effectiveness of rethinking through controlled sampling studies, analyses across task difficulty, and experiments that freeze the main agent. By examining fork placement, advice diversity, and the roles being optimized, these studies characterize the benefits and limitations of rethinking and suggest that its value depends on both the consultation decision and the agent's subsequent behavior.

2 RELATED WORK

Self-correction and the role of feedback. Self-correction methods differ in the feedback available during revision and the training used to support it. Self-Refine uses a single LLM to generate, critique, and iteratively refine an output without additional training (Madaan et al., 2023). However, studies of reasoning and planning find that prompted self-critique can fail to improve accuracy or even revise correct solutions into incorrect ones (Huang et al., 2023; Stechly et al., 2025). Tyen et al. (2024) further distinguish locating an error from repairing it: providing the error location substantially improves correction, suggesting that diagnosis can be a bottleneck. These findings motivate careful attention to the source and reliability of feedback (Kamoi et al., 2024). CRITIC grounds revision in feedback obtained through external tools (Gou et al., 2024), while SCoRe uses multi-turn reinforcement learning to acquire self-correction that operates without external feedback at inference time (Kumar et al., 2025). Together, these studies establish that the effectiveness of correction depends on both its information conditions and how the behavior is learned. Our work examines these conditions within an ongoing agent trajectory, where accumulated observations and additional read-only checks can support reconsideration of earlier decisions.

Reflection and recovery in interactive agents. ReAct interleaves reasoning, actions, and observations, enabling agents to adapt their plans to environmental feedback (Yao et al., 2023). Reflexion converts task feedback into textual reflections retained in episodic memory for subsequent trials (Shinn et al., 2023). More recent work learns recovery within agent trajectories: Agent-R uses tree search to construct revision trajectories and iterative self-training to teach agents to identify errors and resume productive interaction (Yuan et al., 2025). ReGRPO jointly optimizes reflection and corrective actions in local trajectories, with a reflection cost that discourages unnecessary deliberation (Zhang & Shou, 2026). Our focus is the context and control structure of retrospective reasoning. With RETHINK, the acting agent chooses when to request advice, which history window to expose, and what question to ask. A helper instantiated from the same model processes this selected history in a fresh context, can verify its hypotheses through read-only environment access, and returns ad-

vice to the ongoing rollout. This interface makes context selection an explicit part of the correction policy and allows us to study when delegated reconsideration improves on direct continuation.

Credit assignment for agentic reinforcement learning. Outcome-based RL has been applied to tool-augmented retrieval in Search-R1 (Jin et al., 2025). RAGEN studies multi-turn RL and finds that coarse outcome rewards can still lead to shallow agent behavior (Wang et al., 2025). With terminal-reward GRPO (Shao et al., 2024), a shared trajectory advantage provides limited resolution for distinguishing the decisions that lead to an outcome. GiGPO addresses this by grouping actions at recurring environment states to derive step-level advantages alongside episode-level advantages (Feng et al., 2025). Our method constructs comparison groups specifically around an optional RETHINK call. From a shared trajectory prefix, we sample continuations without the call, alternative helper advice, and multiple continuations under each piece of advice. The resulting terminal rewards define distinct relative advantages for invocation, advice generation, and subsequent action (Section 3.3). This construction complements state-based credit assignment by explicitly comparing the value of requesting advice with that of continuing directly, while also learning which advice and continuations are effective. It provides a training mechanism for selective reconsideration and an experimental framework for examining its downstream value.

3 METHOD

We use RETHINK to study whether an agent can improve an ongoing trajectory by reconsidering selected execution evidence in a fresh context. The training procedure gives separate credit to requesting advice, producing it, and acting on it, while retaining task-level supervision for the main agent.

3.1 RETHINKING THROUGH A FRESH CONTEXT

Let x denote a task and $h_t = (x, a_0, o_0, \ldots, a_{t-1}, o_{t-1})$ the interaction history before decision t. The main agent samples $a_t \sim \pi_\theta(\cdot \mid h_t)$ from the environment's native actions augmented with RETHINK. A completed trajectory τ receives an environment-verified task reward $R(\tau)$; the post-training experiments use binary success, $R(\tau) \in \{0, 1\}$.

The tool interface accepts a question q and a requested history length ℓ, serialized in our training scaffold as `RETHINK: ℓ/<question>`. A helper instantiated from the same policy receives a separately constructed context

$$c_t = \mathcal{C}(x_t, q, h_t, \mathcal{D}_t), \qquad g \sim \pi_\theta(\cdot \mid c_t), \tag{1}$$

where x_t is the currently active user request and $\mathcal{D}_t$ contains relevant API definitions. The returned advice g becomes tool output in the main agent's original history. The main agent then continues to execute actions; the helper does not modify the task state or submit an answer on its behalf.

Evidence construction. The current post-training configuration uses a fixed, budgeted evidence selector for $\mathcal{C}$. It retains the latest two completed action–observation pairs, adds relevant earlier records and API definitions when space permits, and separates proposed actions from actions that actually executed. In AppWorld, earlier records are prioritized using recency and dependencies of the proposed code; in BFCL, selection uses the functions in recent calls and the question. The helper input is capped at 8,160 tokens. Although the interface retains the history-length argument, this configuration records it without using it to determine the included records. Thus the experiments train the consultation decision and question under a fixed context-construction rule; they do not establish that the model learns a history-window length.

Information and action boundaries. The helper sees the task, the question, selected execution evidence, and API documentation, rather than the main agent's full reasoning transcript. The default RL helper generates textual advice without executing domain actions. For AppWorld, the controller can supply up to two read-only API-documentation lookups; for BFCL, it supplies definitions already visible to the agent, including those revealed in later user turns. Neither helper receives hidden tests, success labels, or future observations. Tool-enabled helpers used in other experimental configurations are distinct from this default. These prompt and evidence constraints (Appendix D) change the helper's conditioning, but do not guarantee an independent or correct judgment.

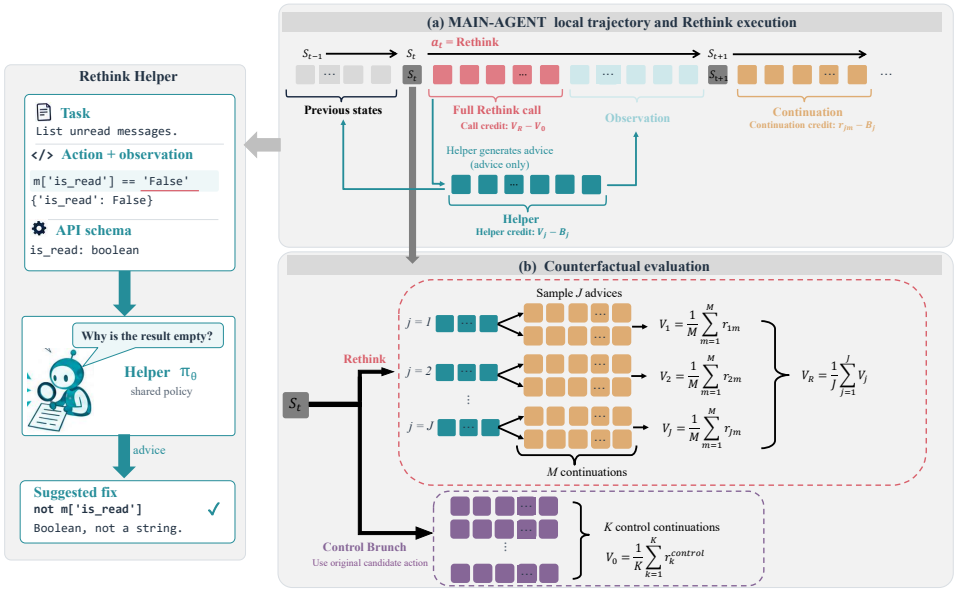


Figure 1: Fresh-context RETHINK and training forks. (a) A helper with shared weights examines selected execution evidence and returns advice. (b) Each call state branches into K direct controls and J advice samples with M continuations each; terminal rewards provide separate credit for invocation, advice, and continuation.

3.2 INITIALIZATION AND ROLLOUT CONSTRUCTION

Before RL, we apply a lightweight SFT warm-up to the base Qwen3-8B model (Appendix A). The main-agent and helper roles then share weights and are optimized jointly from the first RL update.

For each task, we sample N root trajectories. At a naturally sampled RETHINK call, we retain the environment state immediately before the call and construct a tree with K direct continuations, J helper responses, and M continuations per response (Figure 1). Direct continuations restore the pre-call state and sample a new main-agent action, with RETHINK disabled throughout the remaining rollout. Advice branches keep the question and helper context fixed. Each response g_j is returned to a copy of the main history, and its M continuations receive terminal rewards r_{jm}. We reuse the naturally sampled advice and its original continuation as one advice–continuation pair. Branches inherit the remaining environment budget; requesting advice consumes a main-agent decision step, whereas a direct branch does not spend that step on consultation.

We also construct auxiliary trees at selected current or replayed training states to expose the helper to states beyond naturally occurring calls. Their advice and continuations are generated by the current policy. A forced request or replayed prefix receives no invocation-policy gradient, because the current policy did not sample that request at that state. Replay and state selection are summarized in Appendix A. All tree expansion is for training; ordinary inference makes a single helper request per call.

3.3 CREDIT ASSIGNMENT FROM TERMINAL OUTCOMES

Task-level credit. Let $r_i = R(\tau_i)$ be the terminal reward of root trajectory i for task x. We use a leave-one-out numerator normalized by the sample standard deviation s_x of the N root rewards:

$$A_i^{\text{root}} = \frac{r_i - \frac{1}{N-1}\sum_{i' \neq i} r_{i'}}{s_x + \epsilon}, \qquad \epsilon = 10^{-6}. \tag{2}$$

This task-group signal is assigned to all sampled main-agent tokens in the original root trajectory, including call and post-advice action tokens. Constant-reward groups have zero root advantage.

Direct-policy and advice values. For a tree b, define the value of advice j and the mean value of requesting advice by

$$V_{bj} = \frac{1}{M}\sum_{m=1}^{M} r_{bjm}, \qquad V_b^R = \frac{1}{J}\sum_{j=1}^{J} V_{bj}. \tag{3}$$

The direct-policy baseline $V_{0,b}$ is the mean return of all eligible no-RETHINK roots and control branches for the same task in the current batch. A trajectory enters this pool only if its full history contains no helper request. If fewer than two eligible returns are available, we use the tree's K local controls instead. The pooled baseline compares against direct execution on the same task, but can mix different prefixes. It is therefore not an unbiased estimate of the effect of calling at a particular state; the local fallback also permits earlier calls in the inherited prefix.

Invocation credit. For a naturally sampled, valid call, the additional gate advantage is

$$A_b^{\text{gate}} = V_b^R - V_{0,b} + \eta. \tag{4}$$

We apply it to the tokens of the executed RETHINK request, including its arguments. The gate is therefore part of the language-model policy, rather than a separate binary classifier. The constant η encourages consultation and makes the signal positive when $V_b^R > V_{0,b} - \eta$; positivity does not imply that the call improved task success. This bonus affects only the gate update, never the environment scores used in Eq. 2, the control pool, or the advice values.

Advice and continuation credit. For each advice branch, we mix the direct-policy baseline with the leave-one-out mean of the other advice values:

$$B_{bj} = \tfrac{1}{2}V_{0,b} + \tfrac{1}{2(J-1)}\sum_{j'\neq j} V_{bj'}, \tag{5}$$

$$A_{bj}^{\text{helper}} = V_{bj} - B_{bj}, \tag{6}$$

$$A_{bjm}^{\text{cont}} = r_{bjm} - B_{bj}. \tag{7}$$

The helper's own generated tokens receive A_{bj}^{helper} under the helper context; the same advice is masked as tool output in the main trajectory. Continuation credit applies only to sampled actions after that advice. These signals reward both improvement over direct execution and improvement relative to alternative advice. They need not sum to zero within a tree, and receive no further standard-deviation normalization. A reused natural continuation retains its root credit in addition to continuation credit; an extra continuation receives only its continuation credit. Control branches estimate values and receive no additional policy loss.

3.4 JOINT POLICY OPTIMIZATION

We optimize a clipped policy-gradient objective in the group-relative framework of Shao et al. (2024). Each credit channel is normalized by its number of statistical units in the batch: root trajectories, valid natural calls, helper responses, or continuations. For a sampled token u, the effective advantage is the sum of all applicable, normalized credits,

$$\widetilde{A}_u = \sum_{z\in\{\text{root,gate,helper,cont}\}} \frac{1}{|\mathcal{U}_z|}\sum_{v\in\mathcal{U}_z} \mathbf{1}\{u\in v\}A_v^z, \tag{8}$$

where $\mathcal{U}_z$ contains the units for channel z and membership refers only to that unit's trainable token span. Empty channels contribute zero. We detach this sum and apply clipping once per token, rather than clipping each credit separately. With $\rho_u = \pi_\theta(y_u \mid c_u)/\pi_{\text{old}}(y_u \mid c_u)$, the objective is

$$\mathcal{J}(\theta) = \frac{1}{L}\sum_{u\in\mathcal{T}} \min\Big(\rho_u\widetilde{A}_u, \text{clip}(\rho_u, 1-\delta_-, 1+\delta_+)\widetilde{A}_u\Big) - \beta\mathcal{K}_{\text{main}}(\theta), \tag{9}$$

where $\mathcal{T}$ contains generated policy tokens, $L = 2048$ is a fixed token scale, $(\delta_-, \delta_+) = (0.2, 0.28)$, and $\mathcal{K}_{\text{main}}$ is the root-normalized, token-scaled KL penalty to the frozen SFT reference. We set $\beta = 10^{-4}$ for original main trajectories; helper and extra-continuation trajectories have no KL penalty. Environment outputs and copied prefixes are context only. Each update uses the exact generation-time prompt and sampled-token probabilities for every trained action, so changing the training representation does not change its conditioning context. Appendix A gives the remaining details.

4 EXPERIMENTS

Motivated by the prompt-based results in the introduction, we examine how smaller agents learn to make effective use of fresh-context advice. After introducing the setup, we address three questions: **RQ1: Does joint learning improve task success beyond SFT and direct-policy RL?** (Section 4.2); **RQ2: How much can helper learning benefit a fixed main agent?** (Section 4.3); and **RQ3: Does sampling at an execution fork improve coverage over independent sampling?** (Appendices B and C).

4.1 ENVIRONMENTS AND MODELS

Environments. We use AppWorld (Trivedi et al., 2024) and the multi-turn portion of BFCL (Patil et al., 2025), two environments also studied by AgentEvolver (Zhai et al., 2025) and Elastic Horizon (Zhang et al., 2026). Our train/validation partitions are specified below. AppWorld requires agents to complete tasks across stateful applications by executing Python code against their APIs, with success determined by executable task tests. BFCL evaluates function selection and argument generation across changing user requests and environment states. We train on AppWorld's training split and validate on its 57-task development split. For BFCL, we use `multi_turn_base` tasks 0–99 for training and tasks 100–199 for held-out validation. Executions use isolated environment state, and both benchmarks provide binary terminal success rewards.

Models and initialization. Our trainable policy is Qwen3-8B (Yang et al., 2025). For each environment, we initialize it by SFT on demonstrations of main-agent execution and helper advice, covering both consultation and direct execution. The warm-up setup is described in Appendix A. We train Qwen3-8B for two epochs at a learning rate of 10^{-5}. The main agent and helper share model parameters but operate on different contexts: the main agent executes the task, while the helper answers its question using selected execution evidence. As described in Section 3.1, the RL configuration uses a fixed evidence selector and a helper that returns textual advice. Demonstrations are provided by a 27B teacher. Qwen3.8-27B (Qwen Team, 2026) is used for the fork-point sampling study; all methods in the RL comparison use Qwen3-8B.

Training and evaluation. Our joint RL configuration uses 32 tasks per update, four root trajectories per task, and 100 optimizer updates. Each training tree includes two direct controls, four advice samples, and two continuations per advice. The learning rate is 10^{-6} and the training temperature is 0.9; the remaining settings appear in Appendix A. Episodes allow up to 30 main-agent decisions and at most two RETHINK calls. Validation uses eight greedy trajectories per task. We report mean@8, the average success across these trajectories, and pass@8, the fraction of tasks solved at least once. The latter measures task coverage over repeated greedy runs, rather than coverage over eight independent stochastic samples. Evaluation uses ordinary task execution with no counterfactual tree expansion; each RETHINK call returns one advice sample. We select each RL checkpoint by mean@8 among its available evaluations and report both metrics from that checkpoint.

4.2 JOINT REINFORCEMENT LEARNING

We compare four methods initialized from the same task-specific SFT checkpoint. **SFT** evaluates the warm-up model both with and without RETHINK. **GRPO** (Shao et al., 2024) optimizes the main agent using task-group outcome advantages. **GiGPO** (Feng et al., 2025) additionally uses advantages from repeated agent-visible states. Both RL baselines execute tasks without a helper. **RETHINK (ours)** jointly trains task execution, the consultation request, advice generation, and subsequent actions using the credit assignment in Section 3.3. The two SFT evaluations test whether access to the tool is already beneficial after demonstration learning; the RL comparisons assess performance after learning from task outcomes.

Performance and sources of improvement. SFT alone does not make consultation consistently beneficial across environments (Table 2). Enabling RETHINK after SFT gives a small improvement on AppWorld: mean@8 rises from 53.95% to 55.04%, and pass@8 from 77.19% to 78.95%. On BFCL, however, mean@8 falls from 41.38% to 37.00%, and pass@8 from 51.00% to 47.00%.

Table 2: Post-training results on AppWorld dev and held-out BFCL `multi_turn_base`. Scores are percentages; bold highlights our method.

Method	AppWorld (57 tasks)		BFCL (100 tasks)	
	mean@8	pass@8	mean@8	pass@8
SFT w/o RETHINK	53.95	77.19	41.38	51.00
SFT w/ RETHINK	55.04	78.95	37.00	47.00
GRPO	67.11	80.70	46.13	55.00
GiGPO	71.49	87.72	57.50	63.00
RETHINK (ours)	**73.25**	**89.47**	**61.88**	**67.00**

All methods use Qwen3-8B. SFT rows use the initial checkpoint. BFCL SFT w/o RETHINK pools two four-rollout evaluations of that checkpoint, with identical initial prompts and no helper calls.

These mixed results suggest that learning the consultation interface from demonstrations does not necessarily teach *when to invoke it* or *how to use the returned advice* effectively.

Joint RL improves mean@8 over SFT with RETHINK by 18.21 percentage points on AppWorld and 24.88 points on BFCL; the corresponding pass@8 gains are 10.52 and 20.00 points. RETHINK achieves higher scores than both GRPO and GiGPO on both metrics in both environments. Relative to GiGPO, the mean@8 gains are 1.76 points on AppWorld and 4.38 points on BFCL, averaging 3.07 percentage points across the two benchmarks. The pass@8 gains over GiGPO are 1.75 and 4.00 points, respectively. These results support the value of jointly learning to act and advise with credit assigned to the consultation request, advice, and subsequent continuation.

These gains need not arise solely from changing when RETHINK is called. Useful consultation requires the main agent to identify an uncertainty, formulate a question the helper can resolve, and act on the returned advice. A small model may learn the interaction format during SFT while still lacking the task competence needed to perform these steps. Because the two roles share parameters, better task-solving competence may also support better advice: knowing how to execute a task can help the model assess a partial solution and propose a feasible next step. Our objectives train these coupled behaviors from task outcomes. This is a plausible explanation of the RL gains, rather than a component-level attribution from the main table. The frozen-agent study examines helper learning with the main policy held fixed.

Qualitative inspection finds that the prompt-based Qwen3.8-27B model tends to request a final review, whereas the RL-trained 8B model invokes RETHINK at more varied stages. This broader distribution suggests a role for advice during intermediate decisions. The comparison spans different model sizes and training settings, so it does not isolate an RL effect; the goal is useful intervention, rather than uniform invocation positions.

4.3 FROZEN-AGENT REINFORCEMENT LEARNING

Can improving the helper compensate for a fixed main-agent policy? We examine two complementary interventions on AppWorld: training the helper while keeping the 8B main agent fixed, and replacing a self-helper with a larger 27B model. In both cases, the main-agent weights remain unchanged within the comparison. Its rules for selecting consultation states, formulating questions, and acting on advice are therefore not updated, although different advice can change the subsequent trajectory.

Learning only to advise. We train the helper for 100 updates using helper-only GRPO. Eight natural trajectories per training task provide group-normalized outcome advantages for the helper's advice tokens, with no counterfactual expansion or main-agent gradient. Evaluation uses the same 57 development tasks and eight trajectories per task. The helper checkpoint selected by mean@8 achieves 59.21% mean@8 and 78.95% pass@8 (Table 3). Relative to the untrained self-helper in the table, this is a 4.17-percentage-point gain in mean@8, with pass@8 unchanged at 78.95%. Relative to the no-helper configuration, the trained helper improves mean@8 by 5.26 points and pass@8 by 1.76 points, a net increase of one solved task. Thus, advice learning improves average success, but

does not expand task coverage beyond the untrained self-helper at the selected checkpoint. This study uses a different training objective from joint RL and is not a single-variable ablation of main-agent updates. The full learning curves in Appendix A.1 (Figure 2) show that mean@8 and task coverage fluctuate across updates, rather than improving steadily under the fixed main-agent policy.

Table 3: Comparison of helper configurations under a fixed 8B main policy on AppWorld dev, using 57 tasks with eight trajectories per task. Scores are percentages.

Helper configuration	mean@8	pass@8
No helper	53.95	77.19
Self-helper (8B)	55.04	78.95
External helper (27B)	56.80	80.70
Self-helper (8B, w/ RL)	59.21	78.95

Increasing helper capacity. Replacing the untrained 8B self-helper with a 27B helper raises mean@8 from 55.04% to 56.80% and pass@8 from 78.95% to 80.70%. The 27B helper also exceeds the no-helper configuration by 2.85 points in mean@8 and 3.51 points in pass@8. It achieves the highest task coverage among the fixed-main configurations, whereas the RL-trained 8B helper achieves the highest mean@8 (59.21%). Increasing helper capacity can therefore improve end-to-end performance, but the benefits remain modest relative to the joint RL results. Advice must still fit the information supplied in the question and the main agent's ability to translate the response into actions.

Why the consultation policy matters. Together, these comparisons point to the main agent's consultation policy as a key constraint on self-correction. The decision of *when to invoke* RETHINK determines which problem is exposed to the helper, what evidence is available, and how much opportunity remains to recover. A poorly timed request may arrive before there is enough evidence to diagnose the problem, or after the agent has little room to act on a correction. Even at a useful state, the main agent must formulate an actionable question and incorporate the returned advice. Helper capability can translate into task improvement only through this interaction. We therefore treat *when to rethink* as a central learned decision, coupled with learning *how to use advice*, rather than expecting a stronger reviewer alone to solve self-correction. The frozen comparisons are consistent with this account, but do not separately identify timing, question quality, evidence selection, and advice uptake. Joint training addresses these coupled behaviors by assigning credit to the request, the advice, and the resulting continuation.

5 CONCLUSION

We studied self-triggered rethinking in long-horizon LLM agents, using environmental feedback to revisit earlier decisions. We introduced RETHINK, a tool that sends targeted questions and selected execution evidence to a same-model helper in a fresh context, with read-only verification where available. Prompt-based evaluations on Apex Agents and τ^3 show improvements across the evaluated models. Sampling analyses show that advice-seeking branches succeed more often at matched execution prefixes, although branching there does not necessarily increase overall task coverage.

We then developed a branch-sampled reinforcement learning framework that assigns separate credit to invocation, advice generation, and continuation from terminal outcomes. Joint training enables Qwen3-8B to outperform GRPO and GiGPO in average success and task coverage on AppWorld and BFCL. Improving the helper alone also helps, but less than joint training.

These results suggest that self-correction is a learnable interaction between acting and advising. Effective rethinking requires agents to decide when to reconsider, ask evidence-grounded questions, and turn advice into action. It is therefore best treated as part of the agent's decision-making policy rather than a fixed prompting routine.

AI USE DISCLOSURE

We used AI tools primarily to assist with experimental workflows and result analysis, code implementation and debugging, and language polishing where needed. The authors take responsibility for the reported results and the final content of this paper.

REFERENCES

Alibaba Cloud. Qwen3.6-Flash model information, 2026. URL https://www.alibabacloud.com/help/en/model-studio/qwen3-6-flash.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025. URL https://arxiv.org/abs/2506.07982.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023. URL https://arxiv.org/abs/2304.05128.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. *arXiv preprint arXiv:2609.19969*, 2026. URL https://arxiv.org/abs/2609.19969.

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*, volume 38, 2025. doi: 10.52202/085713-1544. URL https://papers.neurips.cc/paper_files/paper/2025/hash/420c9f777c0b4f78d515e53cf74d58b2-Abstract-Conference.html.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.11738.

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. *arXiv preprint arXiv:2310.01798*, 2023. URL https://arxiv.org/abs/2310.01798.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025. URL https://arxiv.org/abs/2503.09516.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024. URL https://arxiv.org/abs/2406.01297.

Kimi Team. Kimi K2.5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026. URL https://arxiv.org/abs/2602.02276.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2409.12917.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023. URL https://arxiv.org/abs/2305.20050.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2308.03688.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023. URL https://arxiv.org/abs/2303.17651.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 48371–48392, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Qwen Team. Qwen3.8-27B model card, 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023. URL https://arxiv.org/abs/2302.04761.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Quan Shi, Alexandra Zytek, Pedram Razavi, Karthik Narasimhan, and Victor Barres. τ-knowledge: Evaluating conversational agents over unstructured knowledge. *arXiv preprint arXiv:2603.04370*, 2026. URL https://arxiv.org/abs/2603.04370.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. URL https://arxiv.org/abs/2303.11366.

Sierra Research. τ^3-bench, 2026. URL https://github.com/sierra-research/tau2-bench.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2402.08115.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/abs/2407.18901.

Gladys Tyen, Hassan Mansoor, Victor Cărbune, Peter Chen, and Tony Mak. Llms cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. URL https://arxiv.org/abs/2311.08516.

Bertie Vidgen, Austin Mann, Abby Fennelly, John Wright Stanly, Lucas Rothman, Marco Burstein, Julien Benchek, David Ostrofsky, Anirudh Ravichandran, Debnil Sur, et al. APEX-agents. *arXiv preprint arXiv:2601.14242*, 2026. URL https://arxiv.org/abs/2601.14242.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022. URL https://arxiv.org/abs/2203.11171.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025. URL https://arxiv.org/abs/2504.20073.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct. *arXiv preprint arXiv:2211.00053*, 2022. URL https://arxiv.org/abs/2211.00053.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

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*, 2023. URL https://arxiv.org/abs/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. URL https://arxiv.org/abs/2406.12045.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-r: Training language model agents to reflect via iterative self-training. *arXiv preprint arXiv:2501.11425*, 2025. URL https://arxiv.org/abs/2501.11425.

Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, Zhaoyang Liu, Bolin Ding, and Jingren Zhou. AgentEvolver: Towards efficient self-evolving agent system. *arXiv preprint arXiv:2511.10395*, 2025. URL https://arxiv.org/abs/2511.10395.

Binjie Zhang and Mike Zheng Shou. Regrpo: Reflection-augmented policy optimization for tool-using agents. *arXiv preprint arXiv:2606.31392*, 2026. URL https://arxiv.org/abs/2606.31392.

Gangyi Zhang, Junjie Meng, Letian Zhang, Wei Wu, Yang Zheng, Dong Wang, Yang Liu, Guanjun Jiang, and Chongming Gao. Elastic Horizon: Discovering the effective interaction frontier in agentic reinforcement learning. *arXiv preprint arXiv:2609.07247*, 2026. URL https://arxiv.org/abs/2609.07247.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. URL https://arxiv.org/abs/2307.13854.

A Training recipe and evaluation details

Training. We use full-parameter SFT for two epochs with AdamW, learning rate 10^{-5}, a cosine schedule, and 5% warm-up. The loss supervises assistant outputs while masking task instructions and tool observations. Joint RL then runs for 100 updates with 32 tasks and four root trajectories per task, learning rate 10^{-6}, and sampling temperature 0.9. Each tree contains two direct controls, four advice samples, and two continuations per advice. The invocation bonus is $\eta = 0.1$; KL regularization uses coefficient 10^{-4} on original main trajectories. Auxiliary states come from current rollouts and a bounded replay pool; branches are regenerated with the current policy after restoring the prefix.

Evaluation. Episodes allow 30 main-agent decisions and at most two RETHINK calls, with output budgets of 4,096 tokens per main-agent response and 8,192 per helper response. Validation is performed every ten updates using eight greedy trajectories per task on AppWorld dev (57 tasks) and held-out BFCL tasks 100–199 (100 tasks). We select checkpoints by mean@8 and report both mean@8 and empirical pass@8 from the same checkpoint, following Section 4.1. No training trees are expanded during evaluation.

A.1 FROZEN-MAIN HELPER-ONLY GRPO TRAINING CURVES

Figure 2 reports the complete validation history of the helper-only GRPO experiment in Section 4.3. The checkpoint selected by mean@8 is update 80, with 59.21% mean@8 and 78.95% best@8. The highest best@8, 82.46%, occurs at update 20, whose mean@8 is 56.58%; the two maxima therefore belong to different checkpoints. At update 100, mean@8 and best@8 are 55.26% and 75.44%, respectively. The curves show that improvements from helper learning are not sustained uniformly across training while the main policy remains frozen.

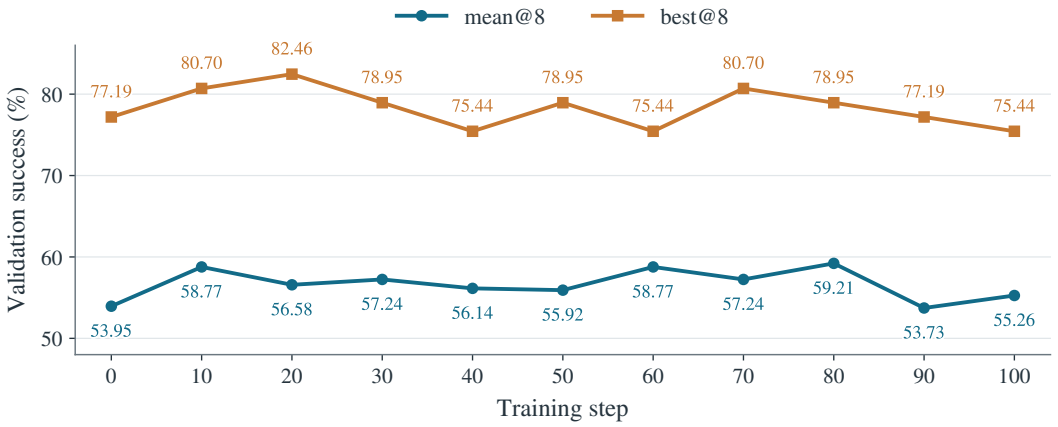
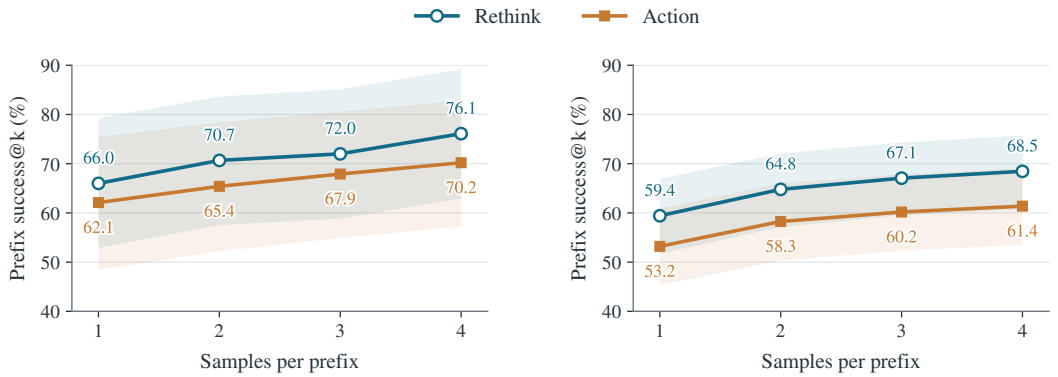


Figure 2: Validation learning curves for frozen-main, helper-only GRPO on AppWorld dev. Each checkpoint is evaluated on 57 tasks with eight greedy trajectories per task. mean@8 averages trajectory success; best@8 is the fraction of tasks solved at least once, equivalent to pass@8 in the main text. All 11 recorded evaluations are shown without smoothing; update 0 precedes RL training.

B SAMPLING AT THE FORK POINT

We study Qwen3.8-27B in AgentEvolver (Zhai et al., 2025) on AppWorld `test_normal` (168 tasks), using temperature 0.9, top-p 1.0, and a 30-decision budget. For each task, we generate six root trajectories. At the first model-initiated RETHINK request, an *action-fork* resamples the next main-agent action six times from the same pre-request prefix. A branch may request advice or continue with code, then execute to completion with the remaining budget. Roots without a request retain their original outcome. A separate baseline generates 36 independent trajectories per task. This design distinguishes the value of consultation at a visited state from the coverage obtained by exploring additional trajectories.



Consultation value across invocation positions. Among 287 prefixes with both scored decision types, requesting RETHINK yields a mean within-prefix success rate of 66.97%, compared with

60.98% for initially direct-code branches: a 5.99-percentage-point difference (95% task-cluster interval $[3.14, 9.10]$). We group these matched prefixes by the decision index at which the fork occurs. The Rethink branches have higher success-rate point estimates in all four groups, while both branch types have lower success at late prefixes. These patterns identify potentially useful consultation opportunities during execution. They describe the states that the model visited and the actions it sampled; they do not establish that moving a request earlier would itself improve the outcome.

Coverage as the sample count increases. We compare expected coverage under subsampling of the recorded candidate banks. For $k = 1, \ldots, 6$, the independent baseline draws $6k$ of its 36 trajectories per task, while action-fork draws k of the six branches available at each forked root. The latter calculation preserves the six-root grouping, and roots that never fork contribute only their original outcome. At the full recorded budget, independent sampling solves 155/168 tasks (92.26%), whereas action-fork solves 140/168 (83.33%). The curve therefore does not show a global coverage advantage from decision-point branching, despite the positive comparison between consultation and direct execution within matched prefixes.

Together, the two views separate a useful local consultation signal from the broader search problem. Sharing a prefix limits exploration of alternative earlier decisions, while the main agent determines which states become consultation opportunities. The positive within-prefix comparison motivates learning when to request advice and how to use it; it does not imply that indiscriminately adding branches will improve coverage. This diagnostic resamples the invocation decision and is distinct from the advice-branch expansion used for training credit assignment.

C WHERE TO FORK: ADVICE-FORK VS. ACTION-FORK

This appendix reports the controlled study that localized the value of rethink-point sampling, which motivated the fork placement in our training procedure (Section 3.3). All runs use the official `simplified_full_code_agent` scaffold with Qwen3.8-27B at temperature 0.9, top-p 1.0, on AppWorld test-normal (168 tasks; baseline pass@36 $= 59.5\%$).

Table 4 summarizes the test-normal comparison.

Operators. We compare two fork operators applied at the first RETHINK-call state of a trajectory:

- **advice-fork**: after the main model issues `RETHINK(n, question)`, fix its decision and sample 6 distinct pieces of advice from the helper, each continued into 6 branches (6 advice $\times$ 6 continuations per root).
- **action-fork**: at the same prefix, re-sample the *main* model 6 times on the decision itself. Some re-samples call RETHINK again (entering the helper pipeline) and others directly produce an action, yielding 6 branches along the action-selection dimension.

Table 4: Fork-position study on AppWorld test-normal (168 tasks; baseline pass@36 $= 59.5\%$). "Oracle" scores the best branch per task over the forked set; NS = not significant. Hard pool = 91 tasks with baseline root-sampling success $\leq 8/36$.

Experiment	Fork scope	Oracle / key metric	vs. baseline
advice-fork v1 (73.6% coverage)	6 root $\times$ 6 advice	56.0% (true) / 52.4% (cons.)	-3.6pp NS / -7.1pp sig. neg.
advice-fork v2 (100% coverage)	+ forced call pre-submit	61.3%	$+1.8$pp, $p = 0.61$
advice-fork boosted	4 candidates + verifier pick	26.4% (hard pool, 91)	$+1.1$pp NS
advice-fork diverse	diversity-injected advice	29.7% (hard pool, 91)	$+4.4$pp NS
action-fork	6 main-model re-samples	50.0%	-9.5pp, $p = 0.00014$ sig. neg.

First-order conclusions from advice-fork. (i) *Coverage comes first.* Moving from v1 to v2 changed only the fork coverage—from 73.6% to 100% by forcing a RETHINK call before submission—and the oracle rose from 56.0% to 61.3% ($p = 0.035$ for the comparison between these variants). This moves the point estimate from below the 59.5% baseline to above it, although v2 does not significantly exceed the baseline ($p = 0.61$). The earlier 8B failure of fork sampling (68.4% vs. 94.7%) was likewise driven by 43% zero-filled roots, i.e., coverage, not advice quality. (ii) *Advice selection and diversity yield inconclusive gains.* On the hard pool, verifier selection

over four advice candidates reaches 26.4%, and diversity injection reaches 29.7%, gains of 1.1 and 4.4 points over their baseline, respectively; neither gain is statistically significant. Diversity injection lowers pairwise code similarity of branches from 0.418 to 0.367, but these results do not establish a reliable success-rate gain or show that advice quality and diversity are irrelevant. (iii) *The difficulty gradient is consistent but underpowered.* On the hard bucket, the three advice-fork variants gain $+3.3/+1.1/+4.4$pp, always in the same direction, while the mid/easy buckets are significantly negative due to prefix-sharing diversity loss. These difficulty-stratified results complement the within-prefix and task-coverage analyses in Appendix B.

Behavioral findings from action-fork. (i) *Consultation decisions vary across samples.* Only 29.5% of roots invoke RETHINK spontaneously; re-sampling the main model 6 times at decision points where it once called yields a re-call rate of only 31.6%, and 68% of decision points produce mixed decisions, indicating that an initial call is often not reproduced when the next decision is sampled again. (ii) *Within-prefix comparison.* Within the 201 mixed decision points, branches that call RETHINK have higher success-rate point estimates by $+2.9$pp (95% CI $[+0.2, +5.9]$, $p = 0.076$), and by $+29.5$pp ($p = 0.031$) in the mid/easy bucket. The pooled direction reverses—a Simpson's-paradox effect, since RETHINK calls are adversely selected into harder states. (iii) *As an exploration operator it is significantly negative overall* (-9.5pp): rethink decision points are adversely selected "stuck" states, and calling consumes step budget.

Takeaway. The fork-position comparisons motivate learning selective consultation, while showing that local gains need not translate into broader task coverage. Under full coverage, advice-fork reaches 61.3% versus the 59.5% baseline, but this difference is not statistically significant. Action-fork has lower overall oracle coverage (50.0%), despite a positive within-prefix comparison for calling RETHINK: $+2.9$ points overall ($p = 0.076$) and $+29.5$ points in the mid/easy bucket ($p = 0.031$). These model-selected branch comparisons do not isolate a causal effect of calling, or establish that advice quality and diversity are unimportant. The hard bucket remains underpowered (91 tasks), and these results are specific to AppWorld test-normal with this scaffold.

Test-challenge check. On 417 AppWorld test-challenge tasks, six roots with six advice branches each solve 177/417 tasks (42.45%), compared with 180/417 (43.17%) for 36 independent baseline trajectories. The paired comparison has 15 fork-only and 18 baseline-only successes ($p = 0.728$), providing no significant evidence of higher overall task coverage from advice forking.

D PROMPT TEMPLATES

We summarize the consultation-specific prompt components for the interface in Section 3.1. They supplement the benchmark's native task instructions and API definitions; task-specific evidence is supplied separately.

Main agent. The main agent first gathers relevant observations, then asks a concrete question about its proposed next action using `RETHINK: n/<question>`. In AppWorld, the request focuses on the selection, calculation, or code needed before a consequential write; in BFCL, it follows at least two completed tool/observation rounds and identifies the proposed function and arguments. The instructions discourage generic planning requests and routine final reviews. After receiving advice, the main agent checks it against visible evidence and executes the chosen action itself. A second consultation is available for a later concrete execution error.

Helper. The helper receives the active task, question, selected completed tool outputs, relevant API definitions, and any proposed action, explicitly marked as unexecuted. It is asked to preserve successful work and give one to three useful next actions, retaining a supported proposal or making the smallest necessary correction. If evidence is missing, it should identify a concrete check rather than invent a result or API. The joint-RL helper returns concise advice, cannot execute domain actions, and cannot request another helper. The request's history-length field remains part of the interface, while the controller uses the fixed evidence-selection rule in Section 3.1.

Warm-up and frozen-main protocol. Warm-up examples pair recorded execution contexts with main-agent or helper responses. The frozen-main experiment in Section 4.3 uses a structured audit

helper: its response gives a verdict, checks task requirements against observed evidence, identifies missing or incorrect work, and proposes one next action. This audit format is distinct from the free-form advice used by the joint-RL helper.