OPTIMIZING WHAT POLICIES LEARN FROM: RECOVERABILITY-AWARE ROLLOUT INTERVENTION LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Critic-free group-based RL has become a scalable paradigm for LLM post-training. However, its effectiveness is constrained by a major limitation: rollouts are allocated uniformly even though their learning value varies substantially across tasks and trajectory states. Despite recent efforts to allocate rollouts adaptively across tasks, existing methods suffer from two core gaps: the **non-stationary gap**, which calls for intervention strategies from learning an evolving policy rather than relying on heuristic rules, and the **non-scalar gap**, which calls for structured control over where and how to intervene rather than merely deciding how many rollouts to generate. To close these gaps, we introduce Recoverability-Aware Intervention Learning (**RAIL**), a training-time framework that turns rollout generation into a learnable intervention process by optimizing structured decisions according to their realized *recoverability gains*. Specifically, RAIL first casts intervention selection as an online contextual-bandit problem and then trains a recoverability controller from intervention traces collected through a shadow-to-live procedure, performing intervention learning while policy evolves. Finally, we evaluate RAIL for its effectiveness, adaptivity, expressiveness, and efficiency, showing consistent gains with constrained rollout budgets. Together, our results establish recoverability-aware intervention as a principled path toward more informative rollout generation, enabling post-training to optimize from stronger and less redundant learning signals.

1 INTRODUCTION

Despite the remarkable success of large-scale pretraining, recent LLM development increasingly relies on post-training to elicit reasoning and task-specific behaviors (Tong et al., 2024; Rafailov et al., 2023). In this regime, progress depends not only on model capacity, but also on how effectively and efficiently training extracts learning signals from model-generated outputs and environment feedback. Critic-free, group-based RL methods such as GRPO (Shao et al., 2024) improve scalability by optimizing policies from relative rewards over multiple sampled responses (Liu et al., 2025;

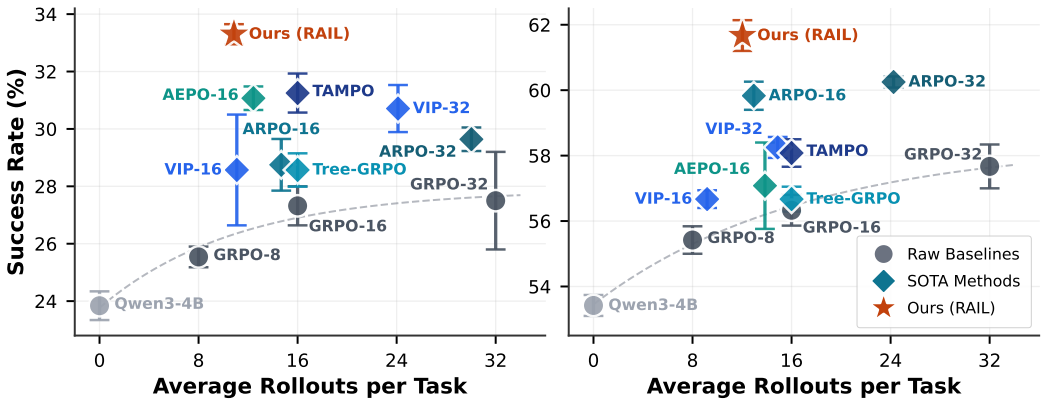


Figure 1: **Performance-rollout Results Preview on AgentBench.** RAIL achieves the highest success rates with comparable or fewer rollouts than uniform GRPO and adaptive intervention baselines on OS task (Left) and DB task(right). More results on further benchmarks and analysis in Section 4.

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

Ahmadian et al., 2024). Yet a key drawback is that they typically allocate rollout budgets uniformly, even though the available learning signal varies substantially across tasks and trajectory states in realistic settings (Dong et al., 2025b). This mismatch is amplified in agentic settings, where rewards are sparse, environment interactions are costly, and early decisions can make later recovery difficult (Dong et al., 2025a). As a result, the marginal value of additional rollouts becomes highly state-dependent: Some states benefit from additional exploration because it reveals reward contrast or alternative successful trajectories (Zheng et al., 2025; Wei et al., 2026), whereas saturated or unrecoverable states only consume rollout budget, weakening optimization by mixing informative and uninformative trajectories in the same update (Wang et al., 2026).

To tackle this problem, prior work has made rollout allocation adaptive from task-level allocation (Yang et al., 2025; Nguyen et al., 2026) to step-level branching (Zheng et al., 2025; Dong et al., 2025b), using signals such as reward variance (Xiong et al., 2025; Fang et al., 2026) or internal signals such as entropy and uncertainty (Wei et al., 2026; Duan et al., 2025). However, they generally suffer from **Two Core Gaps: 1) The non-stationary gap:** existing intervention policies are largely static heuristics, which are only indirect proxies for whether the rollout intervention can expose useful learning signals, and their intervention gain can shift as policies evolve. **2) The non-scalar gap:** prior methods reduce intervention to a scalar budget decision, such as *how many* rollouts to generate, which provides no principled way to coordinate coupled intervention choices, such as *where* or *how* to branch. Thus, a question naturally arises: *Can rollout intervention be learned online from prior realized interventions, while coordinating structured decisions about where and how to intervene?*

To answer this question, we introduce **Recoverability-Aware Intervention Learning (RAIL)**, a training-time framework that learns structured intervention decisions from observed recoverability gains, where recoverability measures how much an intervention improves the reward-distribution signal at a task or trajectory state (Formal definition in Section 2.1). Instead of relying on fixed uncertainty or difficulty proxies, RAIL continually updates a recoverability controller from intervention outcomes and uses it to decide where to branch and select intervention strategies over a structured space. Specifically, RAIL first formulates recoverability learning as an online contextual-bandit problem, where the context is the current trajectory state, the action is a structured intervention, and the observed utility is the realized recoverability gain. It then learns a recoverability controller from intervention traces and deploys it through a structured shadow-to-live procedure: a short shadow phase collects supervision, while a utility-gated live phase selectively applies the controller during policy optimization. This turns rollout generation from a fixed procedure into an adaptive, outcome-driven training loop. Across various benchmarks, RAIL improves **effectiveness, adaptivity, expressiveness, and efficiency** over SOTA intervention baselines. These results show that learning can substantially benefit structured rollout intervention, thereby strengthening policy optimization, pointing toward a broader paradigm for post-training in which both model parameters and the rollout-generation process are adapted to produce higher-value learning signals. Overall, our contributions are:

- **To bridge the non-stationary gap,** we introduce *recoverability* as an outcome-based principle for rollout intervention and propose an online recoverability controller that learns intervention utility from realized reward-distribution gains, rather than relying on fixed uncertainty or difficulty heuristics.
- **To bridge the non-scalar gap,** we formulate rollout intervention over a structured decision space and develop a shadow-to-live deployment procedure that safely collects intervention traces before applying utility-gated online branching during policy optimization.
- We evaluate RAIL along four complementary axes: effectiveness, adaptivity, expressiveness, and efficiency. Across various benchmarks, RAIL improves over SOTA heuristic intervention baselines, highlighting a more informative and principled direction for rollout intervention that provides stronger optimization signals.

2 Rethinking Rollout Intervention through Recoverability

2.1 Preliminaries

GRPO. Group Relative Policy Optimization (GRPO) is a critic-free policy gradient method for reasoning-oriented RL Shao et al. (2024). For each input x, it samples a group of responses $Y = \{y_i\}_{i=1}^{G}$ and estimates the gradient using normalized rewards:

$$\hat{g}(x) = \frac{1}{G}\sum_{i=1}^{G} \hat{A}_i \nabla_\theta \log \pi_\theta(y_i \mid x), \quad \hat{A}_i = \frac{R(y_i) - \mu_R}{\sigma_R + \epsilon}. \tag{1}$$

Uniform Sampling Limitation. While GRPO allocates a fixed rollout budget to each task and computes advantages from uniformly sampled trajectories, the underlying optimization signals are inherently non-uniform: *Some decisions carry richer signals, whereas others contribute little useful information.* This mismatch can waste training compute and dilute policy-gradient updates, motivating *rollout intervention* as an adaptive mechanism for deciding where and how additional rollouts should be generated across tasks and trajectory states.

Rollout Intervention. We define *rollout intervention* as a controlled modification to the default rollout generation process. Given an input x, rather than sampling a fixed group of rollouts from the root-level, it introduces a choice $b \in \mathcal{B}$, where $\mathcal{B}$ denotes a structured intervention space, according to heuristics or learned controllers that determines how the rollout set Y_b is produced. Formally, let z denote the initial or intermediate state of x. We write $Y_b \sim Q_{\theta,b}(\cdot \mid z)$. An intervention may adjust the rollout budget, branch from selected intermediate states, or alter the exploration behavior used to generate continuations under the same policy π_θ. In this view, rollout generation becomes an optimization object in its own right: *instead of passively collecting samples, training can actively shape the rollout distribution toward trajectories that provide stronger learning signal.*

Recoverability. Rollout intervention is beneficial only when additional exploration can expose richer learning signals. For tasks that are already resolved, or states whose future outcomes are overly constrained, further rollouts solely add cost while diluting the optimization signal. In this paper, we introduce recoverability, defined as the intervention gain that remains available at a task or trajectory state. Formally, let z denote a rollout state and $b \in \mathcal{B}$ an intervention choice. We define:

$$\Delta_\theta(z, b) = \mathbb{E}[I(Y_b) - I(Y_\varnothing) \mid z, b, \pi_\theta], \tag{2}$$

where Y_b and $Y_\varnothing$ denote the finite rollout groups generated after applying intervention b and under the default continuation, respectively. $I(Y) = \frac{1}{|Y|}\sum_{y_i \in Y}(R(y_i) - \bar{R}_Y)^2$ measures the reward contrast of a rollout group. A state is recoverable if there exists an intervention that yields positive expected gain, i.e., $\max_{b\in\mathcal{B}} \Delta_\theta(z, b) > 0$. **We instantiate the recoverability signal by finite-group reward contrast**, since in group-based policy optimization with verifiable rewards, reward variance is a principal source of non-degenerate relative-advantage signal (Nguyen et al., 2026); We provide detailed justification of why it can faithfully capture recoverability in Appendix C.1.

Intuitively, recoverability is high when intervention can reveal outcome variation missed by the default rollout process: for example, by expanding under-explored viable continuations, or by branching before early errors accumulate into effectively irreversible failures, analogous to absorbing states in Markov Decision Processes (MDP) Sutton et al. (1998). This concern is amplified in agentic RL, where rewards are sparse, rollouts are costly, and environment interactions can be irreversible.

Task Formulation. We frame rollout intervention as a recoverability-aware decision problem: at every rollout candidate state, choose the intervention with the largest expected recoverability gain:

$$b^*_\theta(z) \in \arg\max_{b\in\mathcal{B}} \Delta_\theta(z, b). \tag{3}$$

Since this gain is not observed before intervention, existing methods approximate it with fixed heuristics such as uncertainty and difficulty. This exposes two core gaps: recoverability is difficult to capture with static heuristics as policy optimization evolves, and effective intervention cannot be reduced to a fixed scalar strategy. We elaborate these two core gaps in the next subsection.

2.2 Challenges in Recoverability-Aware Intervention

The Non-stationary Gap. As discussed above, existing methods allocate additional rollouts using heuristic signals, yet these signals are merely indirect proxies for the intervention gain $\Delta_\theta(z, b)$: a state that appears uncertain may already be unrecoverable, or appears difficult and requires a particular intervention before useful reward contrast can emerge. Moreover, because $\Delta_\theta(z, b)$ depends on the current policy π_θ, its relation to heuristic signals shifts during training. Thus, a fixed heuristic proxy will fail not only because it rarely represents recoverability accurately, but also because the target itself evolves during training. This mismatch can be formulated as below.

Observation 1 (Non-stationary recoverability gap). For a policy π_{θ_t} at step t, define the recoverability oracle as $\mathcal{O}_{\theta_t}(z) := \max_{b \in \mathcal{B}} \Delta_{\theta_t}(z, b)$. For a given intervention strategy f, its cumulative recoverability gap $G_T(f)$ over training can be represented as:

$$G_T(f) = \sum_{t=1}^{T} \mathbb{E}_{z_t} \left[\mathcal{O}_{\theta_t}(z_t) - \Delta_{\theta_t}(z_t, f(z_t)) \right]. \tag{4}$$

A strategy with persistent mismatch from $\mathcal{O}_{\theta_t}$ on a linear number of steps yields $G_T(f) = \Omega(T)$; existing heuristic intervention methods are prone to this case because their fixed proxy-to-intervention rules cannot correct mismatch from realized outcomes. Effective intervention should instead correct such mismatch and achieve sublinear growth.

To bridge this gap, we therefore ask the following two questions. **RQ1 (Effectiveness):** *Does **RAIL** improve final task success over uniform GRPO and heuristic rollout-intervention baselines?* More specifically, **RQ2 (Adaptivity):** *Can **RAIL** track changing recoverability better than static heuristic rules? if so, can we show it both empirically and theoretically?*

The Non-scalar Gap. Another challenge is that effective intervention cannot be fully described by a scalar rollout budget. Once a recoverable state is identified, the intervention must still determine how to reshape the rollout distribution, such as how much budget to spend and how to explore. These choices are coupled, yet prior works either focus mainly on rollout count or treat them largely separate decisions. Thus, the deeper limitation is not only the omission of additional intervention dimensions, but the lack of a principled mechanism for coordinating them. Consequently, such methods can miss recoverability gains that come not from sampling more, but from sampling differently.

Observation 2 (Non-scalar expressiveness gap). Let $\mathcal{B}_{\text{scalar}} \subset \mathcal{B}$ denote a restricted intervention space that varies only one scalar dimension, such as rollout count, while fixing other intervention dimensions; Let $\mathcal{O}_{\mathcal{B}}(z)$ and $\mathcal{O}_{\mathcal{B}_{\text{scalar}}}(z)$ denote the best attainable recoverability gain under the structured and scalar intervention spaces, which induces an expressiveness gap:

$$\mathcal{E}_{\text{scalar}}(z) = \mathcal{O}_{\mathcal{B}}(z) - \mathcal{O}_{\mathcal{B}_{\text{scalar}}}(z). \tag{5}$$

When $\mathcal{E}_{\text{scalar}}(z) > 0$, improving $\mathcal{O}_{\mathcal{B}_{\text{scalar}}}(z)$ alone cannot recover the gain from the structured intervention space. Thus, beyond a single scalar intervention dimension, an effective intervention strategy requires an expressive space that captures coupled rollout intervention decisions.

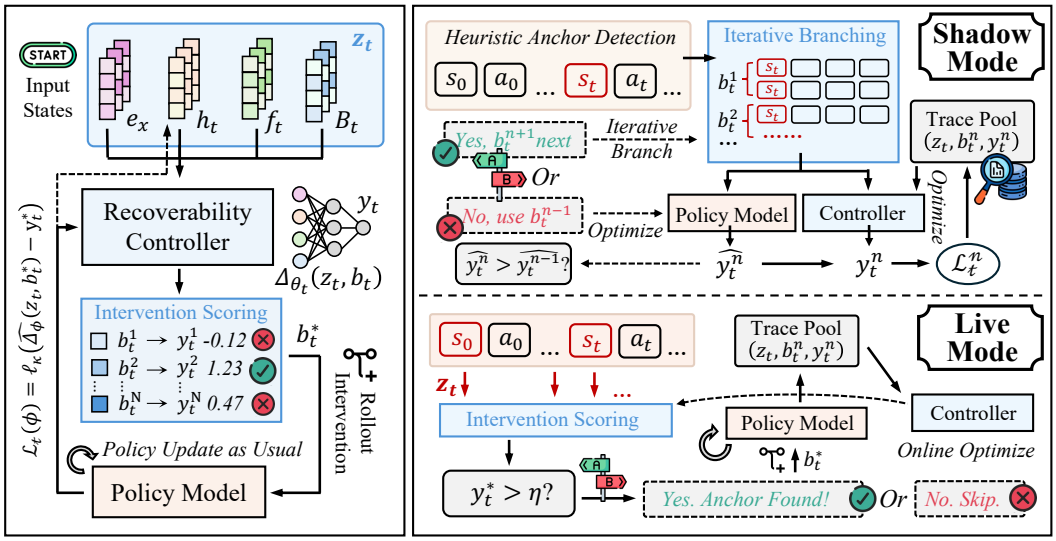
This gap motivates **RQ3 (Expressiveness):** *Does modeling intervention as a structured decision space enable **RAIL** to recover useful training signals, and how do intervention dimensions contribute differently to different tasks?* Finally, another important benefit of rollout intervention is its potential to improve training efficiency by while avoiding saturated or unrecoverable ones. This leads to **RQ4 (Efficiency):** *Can **RAIL** use fewer rollouts to achieve efficient policy improvement?*

3 RAIL: Recoverability-Aware Intervention Learning

To address the above challenges, we propose **Recoverability-Aware Intervention Learning (RAIL)**, a training-time framework that enables policy optimization to receive richer rollout signals while improving efficiency by allocating rollout compute according to learned recoverability. As shown in Figure 2, RAIL couples structured rollout intervention with an online recoverability controller, allowing rollout intervention to adapt with the evolving policy rather than relying on fixed uncertainty or difficulty heuristics. Specifically, the method has two components. Section 3.1 (Figure 2.a) formulates recoverability learning as a contextual-bandit problem, trains the controller from intervention traces; Section 3.2 (Figure 2.b) introduces the structured intervention interface and shadow-to-live deployment, enabling trace collection followed by utility-gated intervention during policy optimization.

3.1 Learning Recoverability from Intervention Outcomes

Recoverability Controller. The central challenge in recoverability-aware intervention is that applying intervention b_t at state z_t yields only an empirical gain y_t, which is only observed after acting. Therefore, RAIL addresses this observability gap by maintaining a learned **recoverability controller**, trained from intervention traces (z_t, b_t, y_t). Formally, under the current policy π_{θ_t}, let $\Delta_{\theta_t}(z, b) := \mathbb{E}[y_t \mid z, b, \pi_{\theta_t}]$ denote the expected gain of applying intervention b at state z. After applying b_t



a) Controller Optimization **b) RAIL Overview: Shadow-to-Live Deployment**

Figure 2: **Overall Framework of RAIL.** (a) The recoverability controller predicts intervention gains from state-action traces. (b) RAIL uses a shadow-to-live procedure: shadow mode collects supervision through heuristic iterative branching, while live mode deploys the learned controller for utility-gated rollout intervention and online adaptation as the policy evolves.

at z_t, RAIL observes finite-sample outcome $y_t = \Delta_{\theta_t}(z_t, b_t) + \xi_t, (\mathbb{E}[\xi_t \mid z_t, b_t, \pi_{\theta_t}] = 0)$, once the intervened traces have been generated. These traces induce a contextual-bandit interaction: at each decision point, the controller observes z_t, chooses one intervention $b_t \in \mathcal{B}$, and observes only the outcome y_t of the selected intervention. Since the gains of unchosen interventions remain counterfactual, the controller must infer intervention utilities from partial feedback accumulated over training. To make this partial-feedback problem learnable, the controller is parameterized by a score function $\widehat{\Delta}_{\phi_t}(z, b)$, where larger scores indicate larger predicted recoverability gain, and trained by fitting its predicted scores to the observed gains:

$$\mathcal{L}_t(\phi) = \sum_{j \in \mathcal{I}_t} w_{t-j}\, \ell_\kappa \left(\widehat{\Delta}_\phi(z_j, b_j) - y_j \right), \quad \mathcal{I}_t = \{ j : \max(1, t - W + 1) \le j \le t \}. \tag{6}$$

Here, $\mathcal{I}_t$ denotes the recent trace window of size W, and w_{t-j} is a non-increasing recency weight that assigns larger weight to more recent traces. The Huber loss ℓ_κ handles noisy empirical gains from finite rollout groups, while the windowed recency weighting helps the controller track the changing recoverability function as π_{θ_t} evolves. We defer the details of the implementation, including contents of state z_t, intervention spaces $\mathcal{B}$, exploration schedules and other details to Appendix B.3.

Recoverability Regret. Having specified how the controller is learned, we next ask if it can avoid the linear growth of the recoverability gap incurred by fixed heuristic strategies. Since $\mathcal{B}$ parameterizes executable interventions, we focus on regret analysis over rounds in which the utility gate triggers intervention. We define the cumulative *recoverability regret* as

$$\mathrm{Reg}_T = \sum_{t=1}^{T} \mathbb{E}_{z_t} \left[\max_{b \in \mathcal{B}} \Delta_{\theta_t}(z_t, b) - \Delta_{\theta_t}(z_t, b_t) \right]. \tag{7}$$

This is the learned-controller version of the recoverability gap in Observation 1: it compares the controller's selected intervention with the best executable intervention in $\mathcal{B}$ at each intervened round. We therefore aim to show that this regret can grow sublinearly under standard conditions.

Theorem 3.1 (Recoverability regret under tracking bound condition)**.** *Given a finite intervention space $\mathcal{B}$ and bounded recoverability gains, let D_T denote the cumulative policy drift variation of Δ_{θ_t}, and A_T denote the cumulative approximation error of the controller class. Under finite-action contextual-bandit feedback with sufficient intervention exploration, the regret of the controller satisfies*

$$\mathrm{Reg}_T \le \tilde{O}\left(\sqrt{T|\mathcal{B}|} + D_T + A_T \right). \tag{8}$$

Consequently, when $D_T + A_T = o(T)$, the controller achieves sublinear recoverability regret.

The terms D_T and A_T generally align with the same stability and learnability conditions under which policy optimization is expected to make progress. Under these conditions, the controller

achieves sublinear regret, so its average recoverability regret decreases asymptotically. A fixed heuristic, however, cannot use outcome feedback to correct proxy mismatch. Thus, even with those assumptions, a persistent per-step mismatch accumulates into linear recoverability regret, which can bias policy optimization toward less informative rollouts, ultimately harming policy optimization. We provide the full assumptions and proof in Appendix C.2.

3.2 Structured Intervention and Shadow-to-Live Deployment

Overview. Although the controller scores interventions by $\widehat{\Delta}_{\phi_t}(z, b)$, these scores are not reliable until the controller has observed enough intervention outcomes. RAIL therefore uses a **shadow-to-live procedure**: a warm-up policy executes structured interventions and records realized gains, allowing the controller to learn from (z_t, b_t, y_t) before it controls live rollout allocation. Once deployed, the controller selects interventions directly and continues updating from new traces as the policy evolves.

Shadow Phase and Structured Trace Collection. In the shadow phase, RAIL uses a lightweight heuristic warm-up policy to collect intervention traces before live deployment, sampling states where recoverability is plausible and exposing recoverability signals for the controller to learn. Given an initial rollout group, the warm-up policy identifies candidate branch points using high-percentile decision entropy, since such anchors mark states where the policy has not collapsed to a single continuation mode and additional branching is more likely to reveal useful reward contrast. At each selected anchor, the warm-up policy considers a finite executable intervention space

$$\mathcal{B} = \mathcal{M} \times \mathcal{T}, \tag{9}$$

where $\mathcal{M}$ is the combination of a discrete set of branch budgets and $\mathcal{T}$ is a discrete set of decoding regimes (e.g. sampling temperatures). This is a minimal instantiation of RAIL's structured intervention interface: the branch budget controls how many additional continuations are generated, while the decoding regimes controls how exploratory those continuations are.

Specifically, at each selected anchor, the warm-up policy applies candidate branches in increasing budget order. It starts with a small branch budget; if the added continuations increase reward variance, the branch is retained and the next larger budget is tried. Otherwise, or once the maximum budget is reached, branching at that anchor stops. **This iterative accept-or-stop design provides two advantages.** First, it searches for a near-best useful branch budget under the current policy, rather than committing to a fixed amount of extra rollout compute. Second, each accepted or rejected trial becomes a separate intervention outcome, giving the controller multiple within-task training signals rather than a single label from the final rollout group. Alongside budget selection, the warm-up policy also chooses the decoding regime for each trial based on task difficulty and current rollout outcomes. Harder or unresolved tasks are assigned more exploratory decoding regimes, since greater continuation diversity is more likely to uncover alternative successful trajectories. We include the specific branch budgets, decoding regimes and the stopping criteria of shadow phase in Appendix B.3.

For each attempted intervention $b_t = (m_t, \tau_t)$ at state z_t, the controller predicts its utility $\widehat{\Delta}_{\phi_t}(z_t, b_t)$ before the branch outcome is observed. After the branch is evaluated, RAIL computes the realized recoverability gain y_t and stores the trace (z_t, b_t, y_t) in the training buffer. The controller is then updated by fitting equation 6 on this buffer, where older traces receive smaller weights. Thus, the shadow phase provides the initial supervised data to train the controller over the same structured intervention space used in live phase, while keeping early rollout allocation under the warm-up policy.

Live Deployment with Utility Gating. After the shadow phase, the controller takes over intervention selection. At each step with rollout state z_t, it scores all executable interventions and selects the one with the largest predicted recoverability gain: $b_t^* \in \arg\max_{b \in \mathcal{B}} \widehat{\Delta}_{\phi_t}(z_t, b)$. To control where and how often rollout branching is triggered, RAIL uses a live utility gate η and executes the selected intervention only when its predicted utility is sufficiently large:

$$b_t = \begin{cases} b_t^*, & \text{if } \widehat{\Delta}_{\phi_t}(z_t, b_t^*) > \eta, \\ \varnothing, & \text{otherwise.} \end{cases} \tag{10}$$

Larger η values make intervention more selective, while smaller values increase branching frequency. When the gate is not passed, RAIL continues the default rollout process without branching. During live training, with probability ρ_t, RAIL instead samples an intervention from an exploration distribution over $\mathcal{B}$ with nonzero support (Detailed in Assumption C.4). Since live deployment only forward-scores a finite set of interventions with a lightweight side predictor, it adds negligible overhead

Method	AgentBench OS	AgentBench DB				WebShop			ToolQA Coffee	
	Overall SR	Overall SR	SELECT	INSERT	UPDATE	Overall SR	L1	L2	Overall SR	Hard
Direct Reasoning										
Qwen3-4B	23.84±0.50	53.42±0.28	42.25±0.50	40.82±0.00	78.79±0.82	33.12±0.57	37.45±0.77	28.89±0.69	74.75±0.66	55.00±1.22
Qwen3-8B	27.68±0.59	56.25±0.95	43.25±2.49	43.00±0.00	82.50±0.50	21.50±0.33	29.22±0.77	14.20±0.58	75.00±0.61	53.00±1.22
Qwen3-14B	28.04±0.93	54.17±2.51	36.75±3.34	45.75±2.28	80.00±6.56	36.75±0.43	48.66±0.67	25.49±0.80	75.62±0.65	51.25±1.30
Qwen3-32B	27.68±0.93	59.08±0.14	43.50±0.50	47.00±0.71	**86.75±0.43**	39.70±0.99	**49.69±1.14**	30.25±2.44	76.62±1.14	58.50±1.50
GRPO Baselines w/o Rollout Intervention										
GRPO-8	25.54±0.36	55.42±0.36	41.50±1.00	45.50±0.58	79.25±0.50	36.90±0.57	40.64±0.94	33.37±0.69	80.00±0.35	64.00±0.71
GRPO-16	27.32±0.68	56.33±0.41	40.75±1.26	47.00±0.00	81.25±0.50	37.20±0.24	44.75±0.67	30.06±0.32	81.88±0.22	67.75±0.43
GRPO-32	27.50±1.70	57.67±0.58	43.00±2.00	48.50±0.58	81.50±0.58	41.65±1.21	41.15±1.91	42.12±1.08	84.25±1.03	75.00±0.00
Step-level Entropy-guided Branching										
ARPO-16	28.75±0.90	59.83±0.43	45.00±0.82	49.50±0.58	85.00±0.00	36.40±0.42	42.28±0.73	30.84±0.17	84.38±0.22	74.75±0.43
ARPO-32	29.64±0.41	60.25±0.17	45.75±0.50	**52.00±0.00**	83.00±0.00	40.70±0.41	46.09±0.50	35.60±0.65	84.25±0.25	74.50±0.50
AEPO	31.07±0.41	57.08±1.32	44.50±0.58	47.00±2.16	79.75±2.22	41.55±0.95	40.53±1.85	42.51±0.89	83.62±0.22	74.25±0.43
Task-level Allocation & Optimization										
VIP-16	28.57±1.93	56.67±0.27	42.00±0.82	47.00±0.00	81.00±0.00	41.60±0.42	44.55±1.02	38.81±0.17	84.00±0.94	73.75±1.64
VIP-32	30.71±0.82	58.25±0.32	46.50±0.58	48.50±0.58	79.75±0.50	44.25±0.22	46.09±0.41	42.51±0.32	84.50±0.00	74.00±0.00
Tree-GRPO	28.57±0.58	56.67±0.38	42.75±0.96	45.75±0.50	81.50±0.58	38.30±0.41	45.47±0.68	31.52±0.48	83.25±0.75	74.50±0.50
TAMPO	31.25±0.68	58.08±0.42	43.00±0.82	47.25±0.96	84.00±0.00	40.95±0.83	45.06±0.74	37.06±1.21	81.25±0.56	66.50±1.12
RAIL	**33.30±0.35**	**61.67±0.41**	**50.50±1.29**	51.00±0.82	83.50±0.58	**45.85±0.38**	46.19±0.53	**45.53±0.95**	**86.75±0.56**	**78.50±1.12**

Table 1: Main results on agentic reasoning benchmarks. We report success rate (SR) across four benchmarks and fine-grained splits. Baselines are trained with Qwen3-4B-Instruct. Best results are bolded and second-best results are underlined. Numerical suffixes denote the max rollout budget.

relative to language-model rollout generation and environment interaction (Details discussed in Appendix B.3). Together with the shadow phase, this utility-gated rule turns recoverability learning into a practical online intervention mechanism: RAIL first obtains structured supervision safely, then deploys the controller selectively where predicted recoverability justifies additional rollout compute.

Co-evolution with Policy Optimization. RAIL integrates with critic-free policy optimization by shaping the rollout groups from which rewards and relative advantages are computed. The controller intervenes at states predicted to yield richer reward contrast, producing more informative trajectories for the policy update. After each update, the changed policy induces new recoverability patterns, and executed interventions provide fresh traces (z_t, b_t, y_t) for further controller training. Thus, RAIL forms an online co-evolution loop: the controller improves rollout allocation for the policy, while the evolving policy supplies new outcomes for the controller to track.

4 Experiments

4.1 Experiment Setup

Benchmarks. We evaluate RAIL on four benchmarks in agentic reasoning spanning classic tool-interaction types. AgentBench-OS and AgentBench-DB (Liu et al., 2024) evaluate reasoning in interactive operating-system and database environments. WebShop (Yao et al., 2022) evaluates web-based product search and purchase, requiring agents to navigate webpages. ToolQA-Coffee (Zhuang et al., 2023) evaluates tabular data, where the agent uses a Python interpreter to perform data analysis. Across all benchmarks, we report success rate (SR) for the same four consecutive seeds with its average. Detailed dataset descriptions and configurations are provided in Appendix B.1.

Baselines. To align with our scope, we focus on the following baselines related to rollout intervention, including: (i) **GRPO**, without adaptive intervention (Shao et al., 2024); (ii) **step-level branching**, such as ARPO (Dong et al., 2025b) and AEPO (Dong et al., 2025a), which spawn extra rollouts from high-entropy states; (iii) **task-level budget allocation**, e.g., VIP (Nguyen et al., 2026) and TAMPO (Dang et al., 2026) which allocates rollouts or changes decoding temperatures from a predicted per-task success signals; and Tree-GRPO (Ji et al., 2025), which share trajectory prefixes to derive step-level advantages. All methods are evaluated under a unified protocol with Qwen3-4B-Instruct. More implementation and adaptation details are provided in Appendix B.2.

4.2 Experiment Results

RQ1: Effectiveness. We present the main results of RAIL in Table 1. Overall, RAIL consistently outperforms all baselines across four agentic reasoning benchmarks, demonstrating that learning rollout intervention from realized recoverability gains provides more effective optimization signals

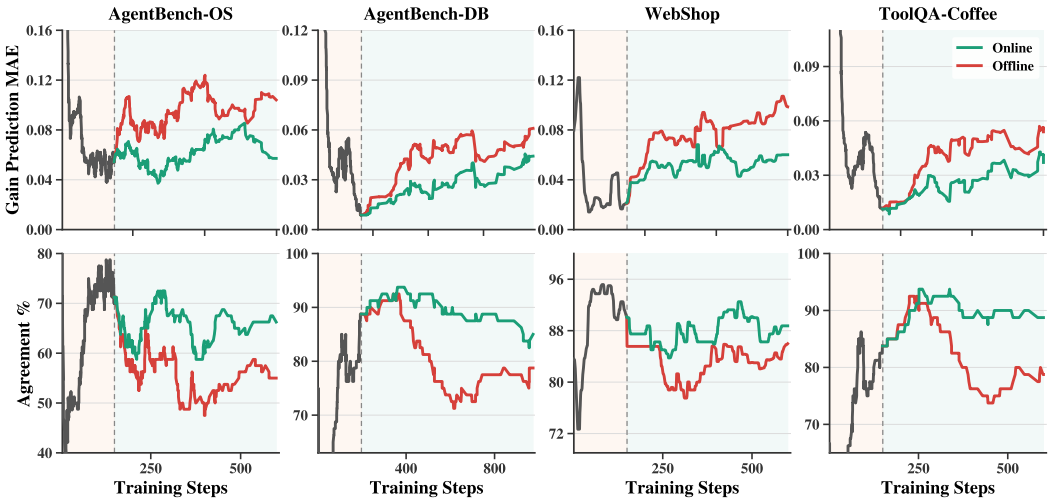


Figure 3: **Adaptivity of the recoverability controller.** In live mode after the shadow phase (black), online updating (green) yields lower MAE and higher sign agreement than a frozen controller (red), indicating that recoverability evolves with the policy and benefits from online tracking.

than uniform sampling or fixed heuristic allocation. Moreover, several insights emerge. First, adaptive rollout intervention is essential for agentic GRPO. Uniform GRPO improves over direct reasoning, but the gains saturate as the rollout budget increases. This indicates that simply increasing rollout count is insufficient. Second, heuristic intervention improves GRPO but is not consistently reliable. While methods such as ARPO and VIP perform strong on some tasks, no heuristic baseline dominates across environments. This supports our motivation that these heuristic rules are only indirect proxies for recoverability, and their utility varies across task types. Third, RAIL gains come from learning where and how to intervene. The gains are particularly clear on difficult splits (e.g. ToolQA-Hard, WebShop-L2), where RAIL improves over the best baseline by a large margin. Overall, these results show that recoverability-aware intervention produces more informative rollout groups, leading to stronger and more consistent policy improvement in agentic RL.

RQ2: Adaptability. To answer RQ2, we compare RAIL's online controller with an offline variant whose controller is trained during the shadow phase but frozen after entering live mode. Figure 3 reports prediction MAE and sign agreement between predicted and realized intervention gains across benchmarks. Averaged over the live phase, online updating reduces MAE and improves sign agreement consistently across benchmarks. Lower MAE indicates more accurate recoverability estimation, while higher sign agreement indicates better judgment of whether an intervention is beneficial. We notice a fixed controller track policy-induced non-stationarity poorly. After entering live mode, the offline controller exhibits larger prediction errors and declining sign agreement, which showcases that recoverability is not a stationary property of a task or state: as the policy changes, the same uncertainty or branching pattern can correspond to different intervention gains, and continuing to update the controller from newly observed intervention traces benefits the adaptation to the evolving policy. **Overall,** these results support RAIL's core goal: rollout intervention should not be fixed, as effective intervention depends on recoverability signals that shift during training.

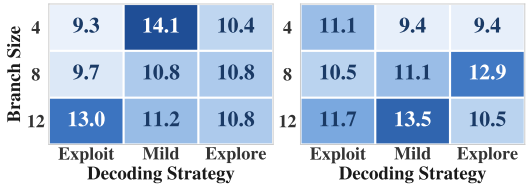


Figure 4: Learned branching preferences across AgentBench-OS (Left) and AgentBench-DB (Right). Each cell reports intervention frequency (%), normalized to 100% per panel. RAIL demonstrate distinct intervention patterns instead of collapsing to a universal choice.

RQ3: Expressiveness. To answer RQ3, we examine how RAIL benefits from modeling rollout intervention as a structured decision. Figure 4 visualizes the empirical intervention distribution selected by the learned controller. RAIL assigns non-trivial probability to multiple branch-size and decoding-regime pairs, rather than collapsing to a single intervention. Meanwhile, the preferred regions differ across environments, which supports our hypothesis that useful rollout intervention is task-dependent and should be learned from realized traces. To further isolate this effect, Table 2 reports controlled ablations where all training settings are kept unchanged except the intervention space or the shadow/live split and we have several insights: First, scalar intervention is insufficient.

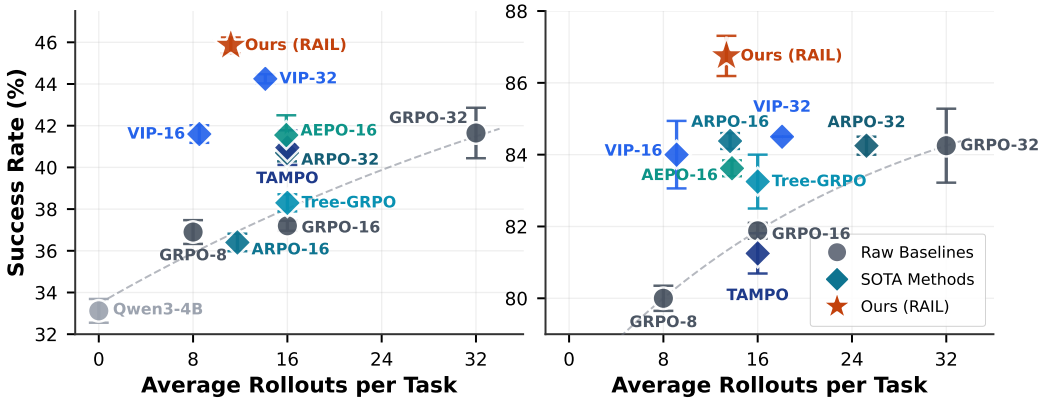


Figure 5: Performance-rollout Results Preview on WebShop (Left) and ToolQA Coffee (Right). RAIL achieves the highest success rates with comparable or fewer rollouts than other intervention baselines.

Method	OS	Δ	DB	Δ
Component Removal				
w/o Branch.	30.89±0.31	↓7.24%	59.67±0.27	↓3.24%
w/o Decod.	31.79±1.07	↓4.53%	59.92±0.17	↓2.84%
Scalar Intervention				
Opt.=2	29.29±1.01	↓12.04%	60.83±0.43	↓1.36%
Opt.=4	34.11±0.78	↑2.43%	59.75±0.50	↓3.11%
Shadow Length				
Shadow 1/3	30.36±1.07	↓8.83%	59.58±0.17	↓3.39%
Shadow 1/2	31.07±0.80	↓6.70%	59.25±0.42	↓3.92%
Dynamic	32.68±0.93	↓1.86%	59.83±0.33	↓2.98%

Table 2: Ablation study on AgentBench.

Method	OS	DB	WebShop	ToolQA
Step-level Branching				
ARPO-16	14.68	12.95	11.76	13.66
ARPO-32	30.04	24.24	25.97	25.23
AEPO	12.43	13.85	15.91	13.82
Task-level Allocation				
VIP-16	11.09	**9.18**	**8.53**	**9.12**
VIP-32	24.11	14.88	14.13	18.07
Tree-GRPO	16	16	16	16
TAMPO	16	16	16	16
RAIL	**10.86**	12.03	11.20	13.35

Table 3: Average rollout across benchmarks.

The scalar variants underperform the full structured space, showing that a non-scalar action space to coordinate benefits our task. Second, intervention-space granularity exhibits a trade-off. Increasing scalar options can help the process, but not uniformly true across all cases. Finally, although longer shadow training provides more initial traces, it also delays live deployment and shortens the period in which intervention learning co-evolves with the policy. **We introduce a variant here with dynamic shadow gate, which we elaborate the design in Appendix B.3.** Overall, these results validate RAIL's core design: effective rollout intervention requires a compact but structured action space, learned online from realized intervention outcomes.

RQ4: Efficiency. Finally, Table 3 reports the average rollout budget used by each method. RAIL uses substantially fewer rollouts than high-budget adaptive baselines. Together with Table 1, Figures 1 and 5 show a clear performance-rollout advantage: RAIL outperforms GRPO-32 across all overall metrics while using less than half of its rollout budget. Compared with adaptive baselines, RAIL also avoids the large budgets used by ARPO-32 and VIP-32 while achieving stronger performance. Overall, these results indicate that recoverability-aware intervention improves efficiency by directing rollout compute toward states where additional exploration is likely to produce useful training signals.

5 Conclusion

We introduced RAIL, a recoverability-aware framework that turns rollout generation from a fixed sampling procedure into a learnable intervention process. By learning from realized intervention gains, RAIL adapts to policy-induced non-stationarity and selects structured interventions over where and how to branch. Across agentic reasoning benchmarks, RAIL consistently improves effectiveness, adaptivity, expressiveness and efficiency over uniform GRPO and heuristic intervention baselines. These results suggest that the rollout-generation process itself should be treated as an optimization object, enabling post-training to learn from stronger and less redundant signals.

AI USE STATEMENT

In this work, we used generative AI tools for helping develop conceptual frameworks, assisting in the writing of proofs, designing and providing feedback on research methodology and experiments, implementing methods, supporting qualitative and thematic data analysis. We have not used generative AI tools for generating synthetic data sets, formulating mathematical related claims, proposing or refining hypotheses, assisting with translation, cleaning and reformatting dataset or interpreting results. Additionally, we used generative AI tools for improving software code and artifacts, draft parts of a research paper, summarizing and refining existing literature as well as editing the research paper to improve readability and helping propose titles for our paper. We have reviewed all AI-assisted work. **Specifically, for the proof part, the deduction is made by human and the proving process is strictly reviewed line by line manually**. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert Schapire. Taming the monster: A fast and simple algorithm for contextual bandits. In *International conference on machine learning*, pp. 1638–1646. PMLR, 2014.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce-style optimization for learning from human feedback in llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12248–12267, 2024.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002a.

Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002b.

Han Bao, Zheyuan Zhang, Pengcheng Jing, Zhengqing Yuan, Kaiwen Shi, and Yanfang Ye. Driftbench: Diagnosing cooperative breakdowns in llm agents under input faults via multi-turn interaction. *arXiv preprint arXiv:2602.02455*, 2026.

Omar Besbes, Yonatan Gur, and Assaf Zeevi. Stochastic multi-armed-bandit problem with non-stationary rewards. *Advances in neural information processing systems*, 27, 2014.

Omar Besbes, Yonatan Gur, and Assaf Zeevi. Non-stationary stochastic optimization. *Operations research*, 63(5):1227–1244, 2015.

Haoran Dang, Cuiling Lan, Hai Wan, Xibin Zhao, and Yan Lu. Temperature as a meta-policy: Adaptive temperature in llm reinforcement learning. *arXiv preprint arXiv:2602.11779*, 2026.

Guanting Dong, Licheng Bao, Zhongyuan Wang, Kangzhi Zhao, Xiaoxi Li, Jiajie Jin, Jinghan Yang, Hangyu Mao, Fuzheng Zhang, Kun Gai, et al. Agentic entropy-balanced policy optimization. *arXiv preprint arXiv:2510.14545*, 2025a.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, et al. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*, 2025b.

Jinhao Duan, James Diffenderfer, Sandeep Madireddy, Tianlong Chen, Bhavya Kailkhura, and Kaidi Xu. Uprop: Investigating the uncertainty propagation of llms in multi-step agentic decision-making. *arXiv preprint arXiv:2506.17419*, 2025.

Yangyi Fang, Jiaye Lin, Xiaoliang Fu, Cong Qin, Haolin Shi, Chaowen Hu, Lu Pan, Ke Zeng, and Xunliang Cai. How to allocate, how to learn? dynamic rollout allocation and advantage modulation for policy optimization. *arXiv preprint arXiv:2602.19208*, 2026.

Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for switching bandit problems. In *International conference on algorithmic learning theory*, pp. 174–188. Springer, 2011.

Yu Han, Kailing Li, Yang Jiao, Yulin Dai, Yuqian Fu, Linhai Zhuo, and Tianwen Qian. 3SPO: State-score-supervised policy optimization for llm agents. *arXiv preprint arXiv:2606.09961*, 2026.

Jiatan Huang, Zheyuan Zhang, Kaiwen Shi, Yanfang Ye, and Chuxu Zhang. Evolverouter: Co-evolving routing and prompt for multi-agent question answering. *arXiv preprint arXiv:2604.05149*, 2026.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for llm agent reinforcement learning. *arXiv preprint arXiv:2509.21240*, 2025.

John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. *Advances in neural information processing systems*, 20, 2007.

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*, volume 2024, pp. 52989–53046, 2024.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362, 2024.

Hieu Trung Nguyen, Bao Nguyen, Wenao Ma, Yuzhi Zhao, Ruifeng She, and Viet Anh Nguyen. Adaptive rollout allocation for online reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2602.01601*, 2026.

Kishan Panaganti, Zhenwen Liang, Wenhao Yu, Haitao Mi, and Dong Yu. Group distributionally robust optimization-driven reinforcement learning for llm reasoning. *arXiv preprint arXiv:2601.19280*, 2026.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Kaiwen Shi, Zheyuan Zhang, and Yanfang Ye. Sage: Answer-conditioned uncertainty targets for verbal uncertainty alignment. *arXiv preprint arXiv:2606.11512*, 2026a.

Kaiwen Shi, Zheyuan Zhang, Zhengqing Yuan, Keerthiram Murugesan, Vincent Galassi, Chuxu Zhang, and Yanfang Ye. Ng-router: Graph-supervised multi-agent collaboration for nutrition question answering. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7508–7527, 2026b.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

Yuxuan Tong, Xiwen Zhang, Rui Wang, Ruidong Wu, and Junxian He. Dart-math: Difficulty-aware rejection tuning for mathematical problem-solving. *Advances in Neural Information Processing Systems*, 37:7821–7846, 2024.

Zihan Wang, Chi Gui, Xing Jin, Qineng Wang, Licheng Liu, Kangrui Wang, Shiqi Chen, Linjie Li, Zhengyuan Yang, Pingyue Zhang, et al. Ragen-2: Reasoning collapse in agentic rl. *arXiv preprint arXiv:2604.06268*, 2026.

Longxuan Wei, Yubo Zhang, Zijiao Zhang, Zhihu Wang, Shiwan Zhao, Tianyu Huang, Huiting Zhao, Chenfei Liu, Shenao Zhang, and Junchi Yan. Entropy-tree: Tree-based decoding with entropy-guided exploration. *arXiv preprint arXiv:2601.15296*, 2026.

Wei Xiong, Chenlu Ye, Baohao Liao, Hanze Dong, Xinxing Xu, Christof Monz, Jiang Bian, Nan Jiang, and Tong Zhang. Reinforce-ada: An adaptive sampling framework under non-linear rl objectives. *arXiv preprint arXiv:2510.04996*, 2025.

Zhicheng Yang, Zhijiang Guo, Yinya Huang, Yongxin Wang, Dongchun Xie, Hanhui Li, Yiwei Wang, Xiaodan Liang, and Jing Tang. Depth-breadth synergy in rlvr: Unlocking llm reasoning gains with adaptive exploration. *arXiv preprint arXiv:2508.13755*, 2025.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Zhiyuan Yao, Yi-Kai Zhang, Yuxin Chen, Yueqing Sun, Zishan Xu, Yu Yang, Tianhao Hu, Qi Gu, Hui Su, and Xunliang Cai. Coba-rl: Capability-oriented budget allocation for reinforcement learning in llms. *arXiv preprint arXiv:2602.03048*, 2026.

Yanfang Ye, Zheyuan Zhang, Tianyi Ma, Zehong Wang, Yiyang Li, Shifu Hou, Weixiang Sun, Kaiwen Shi, Yijun Ma, Wei Song, et al. Llms4all: A review of large language models across academic disciplines. *arXiv preprint arXiv:2509.19580*, 2025.

Yijun Zhang, Fan Xu, Jiaxin Ding, Yule Xie, Shiqing Gao, Xin Ding, Haoxiang Zhang, Luoyi Fu, and Xinbing Wang. Information gain-based rollout policy optimization: An adaptive tree-structured rollout approach for multi-turn llm agents. *arXiv preprint arXiv:2607.06223*, 2026a.

Zheyuan Zhang, Kaiwen Shi, Zhengqing Yuan, Zehong Wang, Tianyi Ma, Keerthiram Murugesan, Vincent Galassi, Chuxu Zhang, and Yanfang Ye. Agentrouter: A knowledge-graph-guided llm router for collaborative multi-agent question answering. *arXiv preprint arXiv:2510.05445*, 2025.

Zheyuan Zhang, Kaiwen Shi, Han Bao, Zehong Wang, Tianyi Ma, and Yanfang Ye. Why semantic entropy fails: Geometry-aware and calibrated uncertainty for policy optimization. *arXiv preprint arXiv:2605.21801*, 2026b.

Yubao Zhao, Weiquan Huang, Sudong Wang, Ruochen Zhao, Chen Chen, Yao Shu, and Chengwei Qin. Training multi-turn search agent via contrastive dynamic branch sampling. *arXiv preprint arXiv:2602.03719*, 2026.

Tianyu Zheng, Tianshun Xing, Qingshui Gu, Taoran Liang, Xingwei Qu, Xin Zhou, Yizhi Li, Zhoufutu Wen, Chenghua Lin, Wenhao Huang, et al. First return, entropy-eliciting explore. *arXiv preprint arXiv:2507.07017*, 2025.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36: 50117–50143, 2023.

Heming Zou, Qi Wang, Yun Qu, Yuhang Jiang, Lizhou Cai, Yixiu Mao, Ru Peng, Xin Xu, Weijie Liu, Kai Yang, Saiyong Yang, and Xiangyang Ji. TRACE: A unified rollout budget allocation framework for efficient agentic reinforcement learning. *arXiv preprint arXiv:2606.11119*, 2026.

APPENDIX CONTENT TABLE

A Related Work **14**

B Implementation Details and Additional Experiments **14**

B.1 Benchmarks 14

B.2 Baselines 15

B.3 Framework Details 16

B.3.1 Structured Intervention Actions 16

B.3.2 State Representation 17

B.3.3 Controller Architecture and Training 18

B.3.4 Shadow-mode Execution 19

B.3.5 Shadow-To-Live Promotion 19

B.3.6 Live Utility Gate 20

B.3.7 Live Exploration 21

B.3.8 Time complexity Analysis 21

C Theory and Proof **21**

C.1 Reward Variance and the Midpoint Recoverability Target 21

C.2 Recoverability Regret Analysis 23

D Discussions **27**

D.1 Intervention-Shaped Sampling and Policy Updates 27

D.2 Recency-Weighted Tracking under Policy Drift 28

D.3 Entropy-Guided Anchor Selection in Shadow Mode 28

D.4 Design Rationale of the Recoverability Controller 29

D.5 Recoverability Beyond Binary Rewards 29

E Prompt Design **29**

A Related Work

Large language models (LLMs) have advanced rapidly in recent years Ye et al. (2025); Zhang et al. (2025). Building on this progress, LLM-driven agents and agentic reinforcement learning have gained prominence for their ability to plan, interact, and solve complex tasks with limited human oversight Zhang et al. (2026b); Shi et al. (2026a;b); Huang et al. (2026); Bao et al. (2026). A central practical challenge accompanying these successes is that RL for LLM reasoning and tool-using agents faces a core inefficiency: rollout budgets are typically allocated uniformly, despite large variation in difficulty and informativeness. Early work addresses this at the *prompt level* by allocating more rollouts to harder or more informative problems, using success rate (Yang et al., 2025; Xiong et al., 2025; Zhao et al., 2026), capability (Yao et al., 2026; Panaganti et al., 2026), or reward variance (Nguyen et al., 2026; Fang et al., 2026) as proxies. While effective, these methods rely on final outcomes and thus overlook where uncertainty arises within multi-step reasoning. This issue is amplified in agentic too use settings, where rewards are sparse and delayed, and failures often originate from specific intermediate decisions, making prompt-level signals insufficient.

To capture finer-grained structure, recent work shifts to *step-level allocation*. However, direct reward signals are harder to retrieve at within steps, motivating the use of *internal signals* such as *Uncertainty*. For example, entropy-based methods identify locally uncertain steps and trigger branching accordingly (Zheng et al., 2025; Wei et al., 2026), while variants further compare uncertainty across steps or between question and tool states to decide whether to expand exploration (Dong et al., 2025b;a). Yet entropy only reflects local ambiguity. To distinguish meaningful from spurious uncertainty, mutual-information-based methods measure dependency between reasoning steps and inputs, capturing whether a branch is informative or merely noisy (Wang et al., 2026; Duan et al., 2025).

Recent concurrent work further learns allocation signals at the state or prefix level. TRACE (Zou et al., 2026) predicts conditional success from prefix histories and allocates visits to promising intermediate anchors. 3SPO (Han et al., 2026) derives state scores from historical success statistics and couples adaptive rollout allocation with step-wise credit assignment and post-step updates. IGRPO (Zhang et al., 2026a) instead expands trajectory trees according to information gain and derives an induced teacher distribution for policy optimization. RAIL differs by learning the realized marginal gain of structured interventions that jointly vary branch budget and decoding behavior, while retaining the underlying group-relative policy objective.

Dynamic rollout allocation also interacts with *policy optimization*. Existing approaches modify token-level advantages (Fang et al., 2026; Dong et al., 2025a), introduce step-level credit assignment (Dong et al., 2025b; Yang et al., 2025; Han et al., 2026), filter low-signal trajectories (Wang et al., 2026), or construct tree-derived rewards and policy targets (Ji et al., 2025; Zhao et al., 2026; Zhang et al., 2026a). These methods often modify rollout collection and the optimization rule jointly. In contrast, RAIL isolates intervention learning at the rollout-generation layer: it retains the critic-free group-relative objective while learning where and how to reshape the rollout distribution from realized intervention outcomes.

B Implementation Details and Additional Experiments

B.1 Benchmarks

AgentBench-OS (Liu et al., 2024) is an operating-system interaction benchmark from AgentBench. The agent receives a natural-language instruction and completes the task by issuing shell commands in an interactive environment. Unlike single-turn reasoning benchmarks, the outcome depends on a sequence of executable decisions, and early commands may alter the environment state in ways that make later recovery easier or harder. This benchmark is therefore suitable for evaluating whether RAIL can identify trajectory states where additional rollout intervention exposes more useful recovery paths. In our paper, We train on 600 bash tasks and evaluate on the full test held-out test tasks that carry a precomputed ground-truth answer. Each episode runs in its own pooled `ubuntu:20.04` container (reset between tasks), and every command is executed under a 5-second hard timeout with truncated output. A task is judged successful either by running its AgentBench checker scripts in the container (all must exit 0) or, when a reference answer is available, by a case-insensitive match with integer-value and bounded-substring tolerance.

Method	Granularity	Calibrated?	Learns When?	Adaptive Strategy?	Strategies
GRPO	Task-level	✗	✗	✗	None
ARPO	Step-level	✗	✗	✗	Branch Count
VIP	Task-level	✓	✓	✗	Branch Count
AEPO	Step-level	✗	✗	✗	Branch Count
Tree-GRPO	Step-level	✗	✗	✗	None
TAMPO	Task-level	✗	✗	✓	Temperature
RAIL	Step-level	✓	✓	✓	Multi-Strategy

Table 4: Cross-dimensional comparison of rollout intervention baselines.

AgentBench-DB (Liu et al., 2024) is a database interaction benchmark from AgentBench. The agent must reason over structured database states and execute operations that satisfy the task instruction. We report both overall success rate and operation-level results over SELECT, INSERT, and UPDATE tasks. This benchmark tests rollout intervention in a structured tool-use setting, where failures may arise from incorrect query construction, wrong operation choice, or insufficient use of intermediate database feedback. In this paper, we train on 1000 tasks and evaluate on the 300-task test split, which is balanced across operation types (100 INSERT, 100 UPDATE, and 100 SELECT-family tasks). The agent uses two tools, `sql_query(query)` and `answer_action(answer)`, against a fresh `MySQL 8` instance whose tables are (re)created and populated per episode. SELECT tasks are graded by a format-tolerant comparison of the returned values (numeric canonicalisation and order-independent set equality); write tasks are graded by *database state*: after the model's writes, the resulting table state is hashed and compared against the state produced by executing the reference SQL on a fresh copy, so any SQL with the correct effect scores as correct regardless of surface form.

WebShop (Yao et al., 2022) is a web-based shopping benchmark in which an agent must satisfy a user request by navigating product search pages and selecting an appropriate item. The agent interacts with the environment through search, page navigation, item inspection, and purchase decisions. We use WebShop to evaluate RAIL under web-navigation rollouts, where the agent must explore multiple candidate products and recover from misleading or incomplete search trajectories. We report overall success rate as well as difficulty-stratified results over L1 and L2 tasks following the difficulty setting from Ma et al. (2024). In this paper, we use the 600-product WebShop index, which exposes 6910 human-instruction goals; goals are partitioned into disjoint index ranges, and we use the full held-out test set as evaluation. The agent has three tools, `search_action(query)`, `click_action(button)`, and `answer_action(answer)`, over at most 10 rounds, and a purchase is realised by clicking `Buy Now`, which terminates the episode.

ToolQA-Coffee (Zhuang et al., 2023) is a tool-mediated question answering benchmark over coffee-domain tabular data. In our configuration, the agent uses a Python interpreter to inspect tables, perform lookup and aggregation, and answer questions that require data analysis rather than memorized knowledge. We evaluate both Easy and Hard splits to measure whether rollout intervention remains useful as the required tool-use and reasoning complexity increases. In our paper, all questions are templated over a single coffee price time series (daily open/high/low/close/volume records). We use a merged split of 600 training for Coffee-hard and 200 test questions. By the official setting, easy questions are single-day lookups and derivations (opening/closing price, percentage change, daily range, bullish/bearish), while hard questions are multi-day aggregations over a date window (extremal price, price range, date of the largest day-over-day move). The agent has two tools, `python_interpreter(code)` and `answer_action(answer)`, over at most 6 rounds; the interpreter runs in a sandboxed worker with a restricted import set and a per-call fresh copy of the data frame. Answers are scored by ToolQA's normalised exact match, augmented with a relative/absolute numeric tolerance so that equivalent numeric formattings (e.g. `147` vs. `147.0`) are accepted.

B.2 BASELINES

GRPO (Shao et al., 2024) serves as the base policy optimization framework without adaptive intervention. It performs group-based updates by normalizing rewards across a fixed number of sampled rollouts, implicitly assuming all rollouts contribute equally regardless of their variability or informativeness.

Step-level Branching. Step-level branching methods, including ARPO (Dong et al., 2025b) and AEPO (Dong et al., 2025a), spawn additional rollouts from intermediate states whose token entropy is high. ARPO tracks the entropy change relative to an initial anchor and branches once it exceeds a threshold, while AEPO splits the global and branch budget through a sigmoid of the question-versus-tool entropy gap and penalizes lineages that branch repeatedly. However, the branch trigger is a fixed function of an output-distribution uncertainty signal, which is decoupled from the environment recoverability that determines whether extra rollouts can restore a useful learning signal.

Task-level Predictive Budget Allocation. These methods adapt a per-task scalar from a predicted or observed reward signal. VIP (Nguyen et al., 2026) relates a group's gradient variance to a Bernoulli success model and solves a budget-constrained program that assigns more rollouts to the tasks predicted to be most variance-rich, whereas TAMPO (Dang et al., 2026) treats the decoding temperature as a reward-driven meta-policy, sampling hotter or cooler decoding where higher reward has been observed. However, both commit to a single scalar per task, how many rollouts, or how exploratory the decoding, and cannot express which kind of intervention a specific mid-trajectory state requires.

Tree-structured Rollouts. Tree-GRPO (Ji et al., 2025) collects rollouts as prefix-sharing trees rather than independent trajectories, computing an intra-tree advantage from the post-branch suffixes and an inter-tree advantage across trees, so that more rollouts fit a fixed budget and a step-level signal is derived from the shared prefixes. However, the tree structure is fixed a priori rather than chosen per state, so compute is not redirected toward the specific decisions whose additional rollouts would most improve informativeness.

B.3 Framework Details

This appendix instantiates the abstract controller interface used in Section 3. The main method requires the controller to compare interventions through $\widehat{\Delta}_\phi(z, b)$; therefore, implementation must specify how actions b are encoded, what pre-intervention information forms the state z, how realized gains y are computed, and how the live utility gate is applied. In this section we describe RAIL algorithm in details.

B.3.1 Structured Intervention Actions

RAIL uses a compact executable intervention space that varies both rollout quantity and rollout behavior:

$$\mathcal{B} = (\mathcal{M} \times \mathcal{T}), \qquad \mathcal{M} = \{4, 8, 12\}, \qquad \mathcal{T} = \{\textsc{Exploit}, \textsc{Mild}, \textsc{Aggressive}\}.$$

Here, each $m \in \mathcal{M}$ denotes the number of additional continuations spawned at an anchor, and each $\tau \in \mathcal{T}$ specifies a decoding regime. For AgentBench-OS and AgentBench-DB, the regimes are

$$\textsc{Exploit} = (T_{\text{base}}, 0.90), \quad \textsc{Mild} = (1.3, 0.98), \quad \textsc{Aggressive} = (1.6, 1.0),$$

where each pair denotes temperature and top-p, and T_{base} inherits the baseline sampling temperature. For WebShop and ToolQA-Coffee, we use more exploratory regimes, with $\textsc{Mild}=(1.8, 0.99)$ and $\textsc{Aggressive}=(2.5, 1.0)$, because these environments require broader search over web or table-operation trajectories.

This 3×3 space gives nine executable branch actions. It is intentionally small enough for online partial-feedback learning, but not tied to these exact values. **The same interface can be extended** by adding another scalar choice, such as prompting style, verifier-guided repair, or rollback actions; or by adding option choices in one scalar, such as what we did in ablation study, where we vary the granularity of the discrete grid: a coarser 2×2 space (branch counts $4, 12$, regimes EXPLOIT, AGGRESSIVE) and a finer 4×4 space that introduces an additional branch budget ($m = 16$) and a fourth intermediate temperature regime. In the controller input, an action is encoded by a four-dimensional vector: one normalized branch-count scalar and a three-dimensional one-hot code for the decoding regime. For the default 3×3 action space (branch counts 4, 8, 12), the branch count is normalized as $m/12$. For action-granularity ablations with different maximum branch budgets, we normalize m by the maximum branch budget in the corresponding action grid. The scalar budget encoding lets nearby branch counts share statistical strength, while the one-hot regime encoding avoids imposing an artificial ordering over qualitatively different sampling behaviors.

Algorithm 1 RAIL: Recoverability-Aware Intervention Learning

Require: Policy π_θ, structured intervention space $\mathcal{B}=\mathcal{M}\times\mathcal{T}$, recoverability controller $\widehat{\Delta}_\phi$, utility threshold η
Ensure: Updated policy π_θ and controller $\widehat{\Delta}_\phi$

Initialize intervention buffer $\mathcal{D}\leftarrow\varnothing$
Initialize mode $q\leftarrow$ SHADOW
for each policy-optimization step t **do**
  Generate an initial rollout group Y_t using π_{θ_t}
  Identify candidate anchors $\mathcal{A}_t$ and construct states z
  if $q=$ SHADOW **then**
    Shadow Mode: Heuristic Trace Collection
    for all $z\in\mathcal{A}_t$ **do**
      for all $b\in\mathcal{B}$ in iterative order **do**
        Execute intervention b and observe $y=I(Y_{\text{after}})-I(Y_{\text{before}})-\lambda C(b)$
        $\mathcal{D}\leftarrow\mathcal{D}\cup\{(z,b,y)\}$
        Update $\widehat{\Delta}_\phi$ from recent traces in $\mathcal{D}$
        if b improves the rollout-group signal **then**
          Retain the generated continuations
        else
          break ▷ accept-or-stop
        end if
      end for
    end for
    if the promotion criterion (fixed or dynamic) is satisfied **then**
      $q\leftarrow$ LIVE
    end if
  else
    Live Mode: Learned Utility-Gated Intervention
    for all $z\in\mathcal{A}_t$ **do**
      Select $b_t^\star\in\arg\max_{b\in\mathcal{B}}\widehat{\Delta}_{\phi_t}(z,b)$
      if $\widehat{\Delta}_{\phi_t}(z,b_t^\star)>\eta$ **then**
        Execute $b_t^\star$ ▷ with exploration during training
        Observe y_t and retain the generated continuations
        $\mathcal{D}\leftarrow\mathcal{D}\cup\{(z,b_t^\star,y_t)\}$
        Update $\widehat{\Delta}_\phi$ from recent traces
      else
        Continue the default rollout process
      end if
    end for
  end if
  Compute rewards and relative advantages from Y_t
  Update $\pi_{\theta_t}\rightarrow\pi_{\theta_{t+1}}$ using the standard critic-free objective
end for
return π_θ and $\widehat{\Delta}_\phi$

B.3.2 STATE REPRESENTATION

The controller must predict intervention gain before the branch outcome is observed, so z_t is restricted to leakage-safe pre-intervention information. We use a 26-dimensional state vector:

$$z_t=\Big[\underbrace{s_t}_{10\text{ scalars}}\ \|\ \underbrace{P_h h_t}_{8\text{ dims}}\ \|\ \underbrace{P_e e_x}_{8\text{ dims}}\Big],\qquad \dim(z_t)=26.$$

The first block s_t contains ten rollout statistics. They are chosen to capture four aspects that determine recoverability from empirical statistical analysis: local uncertainty, trajectory progress, current reward-distribution shape, and diversity of the existing rollout pool. The exact features are listed in Table 5. All statistics are computed from the rollout pool before the attempted branch, together with anchor metadata and budget counts; the after-branch outcome is never used as an input feature.

The last two scalar features, average pool entropy and tool-sequence diversity, were included after offline feature selection: average pool entropy produced the largest act/skip-AUC improvement, and

Feature	Role in recoverability prediction
Anchor entropy	Measures local decision uncertainty; high entropy indicates that the policy has not collapsed to a single continuation mode.
Anchor round	Locates the intervention point in the trajectory; early mistakes often create larger downstream recovery effects.
Anchor round fraction	Normalizes position by the maximum number of rounds, making trajectory progress comparable across tasks.
Budget used fraction	Indicates remaining rollout capacity, which affects whether further branching is worth its cost.
Before-branch reward mean	Captures current task solvability under the existing pool.
Before-branch reward standard deviation	Measures existing reward contrast before intervention.
Distance to 0.5	Estimates the remaining gain ceiling; pools near all-success or all-failure have less relative-reward contrast.
Failed-anchor count	Tracks repeated unsuccessful intervention attempts, signaling potentially low recoverability for the current task.
Average pool entropy	Measures uncertainty across the rollout pool, not only at the selected anchor.
Tool-sequence diversity	Measures behavioral diversity by counting distinct tool-call sequences relative to pool size.

Table 5: Pre-intervention scalar features used by the RAIL controller.

tool-sequence diversity gave an additional gain. A tested visitation-uncertainty feature was removed because its signal largely reflected recurrence artifacts rather than recoverability. The projected blocks provide lightweight context beyond hand-designed statistics: h_t is the policy hidden state at the anchor and e_x is the task embedding. Both are projected to eight dimensions, which keeps the controller cheap and the projection matrices P_h and P_e are trained with the controller, while the policy hidden states are detached so that controller training does not update the underlying policy model.

B.3.3 Controller Architecture and Training

Given the 26-dimensional state and 4-dimensional action code, the controller predicts a scalar gain from a 30-dimensional state-action input. We use a single-output neural network rather than a multi-head action classifier, because each trace observes the outcome of only one selected action. This design matches the contextual-bandit feedback structure: the controller learns a utility function over (z, b) pairs and generalizes across nearby states and actions. The default network uses a 64-dimensional hidden layer with GELU activation, dropout, and a skip connection from the raw state-action input to the output layer. **The controller runs separately from the policy model; hidden states are detached before entering the controller, so controller gradients never update the language model.** We further elaborate our design rationale in Appendix D.4

During the run, each executed branch produces a trace (z_t, b_t, y_t). The controller is trained on a replay buffer with recency-weighted Huber regression:

$$\mathcal{L}(\phi) = \frac{\sum_j w_j \ell_\kappa \left(\widehat{\Delta}_\phi(z_j, b_j) - y_j \right)}{\sum_j w_j}.$$

We use Huber loss because finite rollout groups produce noisy gain labels, and we use recency weighting because recoverability changes as the policy evolves. In implementation, the hard window in equation 6 is realized as an exponential half-life over recent traces.

Recoverability label and intervention cost. Our benchmarks are all trained with binary reward, despite benchmarks such as WebShop provide dense reward options. The implementation computes recoverability gain through the reduction in distance to the balanced reward regime:

$$y_t = [d(Y_{\text{before}}) - d(Y_{\text{after}})] - \lambda C(b_t), \qquad d(Y) = |\bar{R}_Y - 0.5|.$$

This label is monotone with the reward-contrast objective in the binary case: moving the rollout pool toward $\bar{R}_Y = 0.5$ increases the variance term $p(1-p)$ and creates stronger relative-reward signal. In practice, a cost penalty is included in all reported runs with $\lambda = 0.005$. It prevents the controller

from learning a trivial preference for larger branch budgets and makes the predicted gain correspond to utility after accounting for rollout cost.

B.3.4 SHADOW-MODE EXECUTION

The main text describes shadow mode conceptually; here we specify its execution rule. RAIL begins from an initial pool of four rollouts and detects candidate anchors using normalized token entropy. At each anchor, the warm-up policy tries branch budgets in increasing order, $4 \to 8 \to 12$, clamped by the remaining rollout budget (Rollout budget is the maximum budget allowed for branching and allocation.) A branch is retained if the pooled reward distribution moves closer to the balanced regime; otherwise, the sweep at that anchor stops. Each accepted or rejected trial is logged as a separate trace, so one anchor can provide multiple supervised outcomes for the controller. The decoding regime is chosen from the current pool mean: easier pools use EXPLOIT, intermediate pools use MILD, and harder or unresolved pools use AGGRESSIVE; when adaptive decoding is enabled, this regime is recomputed before each budget increment.

B.3.5 SHADOW-TO-LIVE PROMOTION

The shadow phase is only needed to bootstrap the recoverability controller before it is trusted to control live rollout allocation. In practice, we choose a fixed proportion of the total steps to serve as the shadow length for simplicity. We demonstrate how the change affects the training in Table 2 and discussed its trade-offs in the experiment sections. However, a fixed gate is nevertheless a coarse choice: if too short, the controller may enter live mode before its predictions are decision-relevant; if too long, training continues to spend rollout budget on heuristic trace collection after the controller is already useful. **Since controller reliability depends on both the benchmark and the evolving policy, we also evaluate a dynamic promotion rule that switches from shadow to live mode once the controller demonstrates sustained out-of-sample predictive skill.**

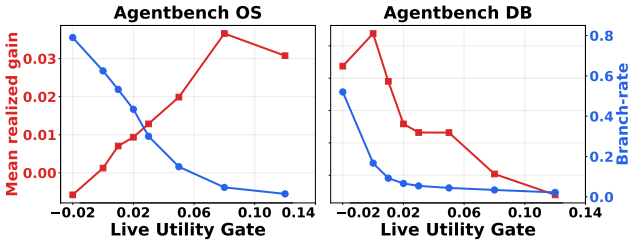


Figure 6: Live Utility Gate Trade-off against realized gain on AgentBench-OS (Left) and AgentBench-DB (Right).

We measure this skill prequentially: before each controller update, we record the prediction made for the executed intervention and compare it with the realized gain after the outcome is observed. This predict-before-update protocol prevents the promotion signal from being inflated by fitting the same trace. We use sign agreement rather than regression error,

$$a_t = \mathbf{1}\left[\operatorname{sign}\left(\widehat{\Delta}_{\phi_t}(z_t, b_t)\right) = \operatorname{sign}(y_t)\right],$$

because live deployment primarily depends on whether an intervention is predicted to improve over the no-op baseline. In contrast, absolute regression error can be misleading when realized gains are concentrated near small negative values: a nearly constant predictor may achieve low MAE while providing little useful act/skip information.

Let $\bar{a}_t$ denote the mean sign agreement over the most recent W intervened anchors. The dynamic rule promotes the controller from shadow to live mode when three conditions hold simultaneously: (i) at least $n_{\min}$ anchors have been observed; (ii) all nine executable branch actions have been observed at least once (During shadow mode, a small coverage component occasionally samples less-used (m, τ) cells so that all executable actions can be observed before live promotion); and (iii) $\bar{a}_t$ exceeds a chance-adjusted threshold for P consecutive anchors. Specifically, we use

$$\beta_{\text{bar}} = 0.5 + 2\sqrt{\frac{0.25}{W}},$$

which is two binomial standard deviations above chance agreement under a balanced-sign null model. Promotion is latched once triggered. As a safety fallback, if the dynamic gate has not fired by a maximum step cap T_{cap}, the controller is promoted by the cap; conversely, if the controller never

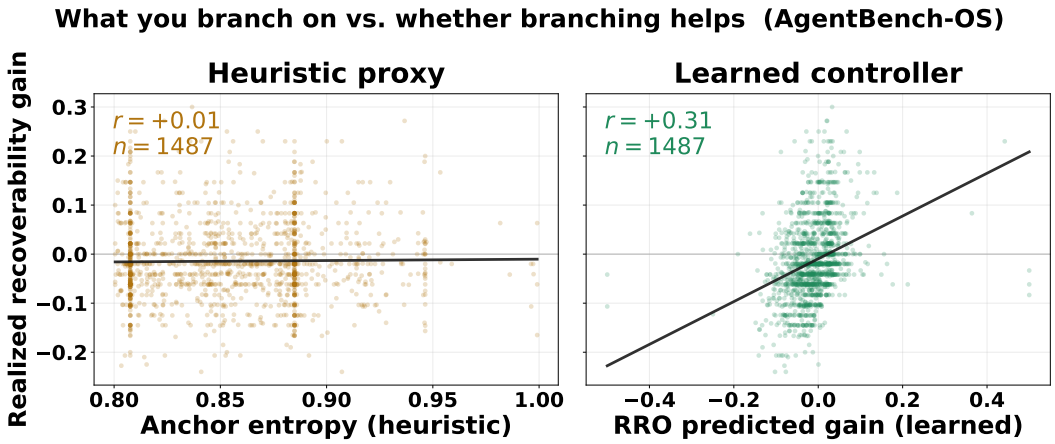


Figure 7: Recoverability is not readable from entropy, but it is learnable.

demonstrates stable predictive skill before the end of training, it remains in shadow mode rather than deploying an unreliable predictor.

In the dynamic-promotion ablation, we set the sign-agreement window to $W = 60$ intervened anchors, the consecutive-success requirement to $P = 60$ anchors, the minimum number of observed anchors to $n_{\min} = 50$, and the safety step cap to $T_{\text{cap}} = 200$ rollout steps, which gives a chance-adjusted promotion threshold of $\beta_{\text{bar}} \approx 0.629$. Here W is the trailing window over which the mean sign agreement $\bar{a}_t$ is computed, P is how many consecutive anchors $\bar{a}_t$ must remain above β_{bar} before promotion, $n_{\min}$ is the warm-up floor of anchors that must be seen first, and T_{cap} is the fallback step at which the controller is promoted regardless. Under this rule, the controller promotes by demonstrated skill rather than by the cap: on AgentBench-OS it enters live mode at step 101 with rolling sign agreement 0.650, and on AgentBench-DB at step 165 with rolling sign agreement 0.783—both well before T_{cap}. Thus, dynamic promotion adapts the shadow length to the controller's observed reliability, reaching live intervention earlier when the controller is ready while avoiding premature deployment when its predictions are not yet stable.

B.3.6 Live Utility Gate

In live mode, the controller scores the nine executable branch actions and compares the best predicted branch against the no-op baseline. The no-op has fixed gain zero, so the live decision rule is

$$b_t^\star = \arg\max_{b \in \mathcal{B} \setminus \{\varnothing\}} \widehat{\Delta}_{\phi_t}(z_t, b), \qquad b_t = \begin{cases} b_t^\star, & \widehat{\Delta}_{\phi_t}(z_t, b_t^\star) > \eta_{\text{gain}}, \\ \varnothing, & \text{otherwise.} \end{cases}$$

Thus, η_{gain} is a margin over the no-op baseline: larger values make RAIL more selective, while smaller values increase branching frequency. We use $\eta_{\text{gain}} = 0.02$ for our experiments. In practice we recommend tuning the threshold within a small range because the reward scale and anchor frequency differ across these environments. Here we provide another ablation study on how the change of η_{gain} affect overall recoverability gain in Figure 6. Unlike shadow mode, live mode executes a single selected branch action rather than running an accept-or-stop sweep.

It's worth noting that Recoverability is not readable from entropy, but it is learnable. Figure 7 probes why rollout intervention must be learned rather than triggered by a fixed uncertainty signal. Both panels are computed on the same set of live-phase decision states logged during RAIL training on AgentBench-OS ($n = 1487$ anchors), and both plot, on the vertical axis, the realized recoverability gain — the change in finite-group reward contrast actually observed after the intervention. The left panel places on the horizontal axis the per-anchor decision entropy, the local-uncertainty signal on which step-level branchers such as ARPO and AEPO base their intervention decisions; it is essentially uncorrelated with realized gain (Pearson $r = +0.01$), indicating that knowing a state is high-entropy tells one almost nothing about whether intervening there will expose additional learning signal. The right panel places on the horizontal axis the gain predicted by RAIL's recoverability controller, which is trained online from realized outcomes; its predictions track the realized gain far more closely ($r = +0.31$).

B.3.7 LIVE EXPLORATION

To maintain coverage of the finite intervention space and guarantee sufficient exploration, live training uses a small exploration probability over executable branch cells. Exploration is directed toward under-covered cells rather than sampled uniformly, which improves coverage of the 3×3 action space under limited live-mode budget. The exploration rate decays over training. We discuss how RAIL enforces sufficient exploration in Appendix C.2.

B.3.8 TIME COMPLEXITY ANALYSIS

While wall-clock time is difficult to compare fairly across implementations because it depends on hardware, serving configuration, batching, parallelism, environment latency, and system load, we instead analyze computational complexity. RAIL leaves the asymptotic cost of the training loop unchanged relative to GRPO. The dominant term for both is language-model rollout generation and environment interaction, which scales as $O(G \cdot L \cdot E)$ for G rollouts of L generated tokens over E environment steps; every other component is lower-order. RAIL's only per-step addition is the recoverability controller, a small side-predictor evaluated at candidate anchor states. Scoring an anchor is a single forward pass of a two-layer MLP ($\sim$2,000 parameters) over the $|\mathcal{B}|$ executable interventions, which is negligible compared with LLM rollout generation. Consequently, the controller contributes no measurable overhead: RAIL's compute is governed by how many rollouts it generates, not by the intervention machinery. In summary, since the utility gate suppresses branching at states with low predicted recoverability, RAIL in fact issues fewer rollouts per task on average than uniform GRPO-32 (Table 3), and it's live time branching cost is comparable to other methods such as ARPO or AEPO despite the added controller.

C THEORY AND PROOF

C.1 REWARD VARIANCE AND THE MIDPOINT RECOVERABILITY TARGET

The recoverability target in this paper is instantiated as an increase in reward variance, which is an indirect proxy for optimization signal quality. This appendix justifies why RAIL uses reward variance as the recoverability signal and why the midpoint expected reward, i.e., success probability $p = 0.5$, is the natural target for informative rollout intervention. The argument is closely related to the variance analysis of VIP (Nguyen et al., 2026), which shows that in group-based RL with verifiable rewards, the prompt-dependent factor in the projected-gradient variance is governed by the binary reward-variance term $p(1-p)$. We adapt this insight from prompt-level rollout allocation to our state- and intervention-level setting: rather than allocating more rollouts to prompts with high predicted variance, RAIL asks whether an intervention increases the reward-distribution contrast available at a task or trajectory state.

Setup. Fix a rollout state z and an intervention $b \in \mathcal{B}$. Let $R_b(z) \in \{0, 1\}$ denote the verifier reward of a rollout generated after applying intervention b. We define

$$p_b(z) = \Pr(R_b(z) = 1), \qquad \mu_b(z) = \mathbb{E}[R_b(z)], \qquad V_b(z) = \mathrm{Var}(R_b(z)). \tag{11}$$

For a group of n conditionally i.i.d. rollouts, let $R_1, \ldots, R_n \sim R_b(z)$ and $\bar{R} = \frac{1}{n}\sum_{i=1}^{n} R_i$. Group-based methods form reward-dependent relative advantages, for example

$$A_i^{\mathrm{cent}} = R_i - \bar{R}, \qquad A_i^{\mathrm{loo}} = R_i - \frac{1}{n-1}\sum_{j \neq i} R_j. \tag{12}$$

Thus, when all rewards in a rollout group are identical, the reward-dependent advantage signal degenerates. Nontrivial reward contrast arises only when the rollout group contains different reward outcomes.

Lemma C.1 (Reward variance controls relative advantage contrast)**.** *For conditionally i.i.d. rewards with variance $\sigma_R^2 = \mathrm{Var}(R_b(z))$, the expected mean squared relative advantages satisfy*

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left(A_i^{\mathrm{cent}}\right)^2\right] = \frac{n-1}{n}\sigma_R^2, \qquad \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left(A_i^{\mathrm{loo}}\right)^2\right] = \frac{n}{n-1}\sigma_R^2. \tag{13}$$

Proof. For the centered estimator,

$$\sum_{i=1}^{n}(R_i - \bar{R})^2 = \sum_{i=1}^{n} R_i^2 - n\bar{R}^2. \tag{14}$$

Let $\mu = \mathbb{E}[R_b(z)]$. Since $\mathbb{E}[R_i^2] = \sigma_R^2 + \mu^2$ and $\mathbb{E}[\bar{R}^2] = \mathrm{Var}(\bar{R}) + \mu^2 = \sigma_R^2/n + \mu^2$, we have

$$\begin{aligned}\mathbb{E}\left[\sum_{i=1}^{n}(R_i - \bar{R})^2\right] &= n(\sigma_R^2 + \mu^2) - n\left(\frac{\sigma_R^2}{n} + \mu^2\right) \\ &= (n-1)\sigma_R^2.\end{aligned} \tag{15}$$

Dividing by n gives the first equality in equation 13. For the leave-one-out estimator,

$$A_i^{\mathrm{loo}} = R_i - \frac{n\bar{R} - R_i}{n-1} = \frac{n}{n-1}(R_i - \bar{R}). \tag{16}$$

Substituting equation 16 into the centered result gives the second equality in equation 13. □

Connection to VIP's projected-gradient analysis. Lemma C.1 gives a direct advantage-level justification for reward variance. VIP provides a complementary gradient-level justification: under standard conditional i.i.d. rollout assumptions and second-order decorrelation assumptions between rewards and projected score terms, the per-prompt projected-gradient variance of Dr. GRPO and RLOO is proportional to the same reward-variance factor (Nguyen et al., 2026). In the notation of VIP, for binary rewards encoded as $\{-1, +1\}$ with success probability p, the reward-dependent term is $4p(1-p)$; for the $\{0, 1\}$ encoding used here, the corresponding term is $p(1-p)$. The two encodings differ only by a constant factor and therefore have the same maximizer. This establishes that reward variance is not merely a heuristic measure of diversity: it is the reward-dependent contrast term that governs relative-advantage signal, and appears explicitly in the projected-gradient variance analysis of group-based RL.

Proposition C.2 (Midpoint maximizes binary reward contrast)**.** *For verifier rewards $R_b(z) \in \{0, 1\}$,*

$$V_b(z) = \mathrm{Var}(R_b(z)) = p_b(z)(1 - p_b(z)). \tag{17}$$

This quantity is uniquely maximized at $p_b(z) = 1/2$, equivalently $\mathbb{E}[R_b(z)] = 0.5$.

Proof. Since $R_b(z)$ is Bernoulli with success probability $p_b(z)$,

$$\mathrm{Var}(R_b(z)) = \mathbb{E}[R_b(z)^2] - \mathbb{E}[R_b(z)]^2 = p_b(z) - p_b(z)^2 = p_b(z)(1 - p_b(z)). \tag{18}$$

Moreover,

$$\frac{d}{dp_b}p_b(1-p_b) = 1 - 2p_b, \qquad \frac{d^2}{dp_b^2}p_b(1-p_b) = -2 < 0. \tag{19}$$

Thus the unique maximizer is $p_b = 1/2$. The variance is zero at $p_b \in \{0, 1\}$, corresponding to rollout groups that are either all failures or all successes. □

Recoverability gain. Proposition C.2 shows that binary reward contrast is maximized at the midpoint and vanishes at both saturated extremes. Therefore, rollouts with $p_b(z) \approx 0$ are effectively unrecoverable under intervention b, rollouts with $p_b(z) \approx 1$ are already solved, and rollouts near $p_b(z) = 0.5$ provide the richest expected reward contrast. This motivates measuring recoverability by how much an intervention improves the reward contrast of the resulting rollout group.

In the population view above, this contrast is captured by the reward variance $V_b(z) = \mathrm{Var}(R_b(z))$. In the actual finite-group setting used by RAIL, we estimate the same signal through the empirical reward contrast

$$I(Y) = \frac{1}{|Y|}\sum_{y_i \in Y}\left(R(y_i) - \bar{R}_Y\right)^2. \tag{20}$$

Thus, for a rollout state z and intervention $b \in \mathcal{B}$, RAIL defines recoverability as the expected intervention-induced gain in finite-group reward contrast:

$$\Delta_\theta(z, b) = \mathbb{E}[I(Y_b) - I(Y_\varnothing) \mid z, b, \pi_\theta], \tag{21}$$

where Y_b denotes the finite rollout group generated after applying intervention b, and $Y_\varnothing$ denotes the rollout group generated under default continuation. A positive gain $\Delta_\theta(z, b) > 0$ indicates that the intervention exposes additional relative reward signal beyond default rollout generation. Conversely, $\Delta_\theta(z, b) \leq 0$ indicates that the intervention does not improve the reward-distribution signal and should not be preferred unless compensated by another objective, such as exploration diversity or downstream verifier information.

Takeaway. The midpoint target is not an arbitrary heuristic. For binary verifiable rewards, $p = 0.5$ is the unique point at which the rollout distribution maximizes learning signals. Since group-based policy optimization relies on relative reward differences, these reward contrast provides a principled proxy for the useful reward-dependent signal available to the update. RAIL therefore instantiates recoverability as the intervention-induced improvement in finite-group reward contrast:

$$\Delta_\theta(z, b) = \mathbb{E}[I(Y_b) - I(Y_\varnothing) \mid z, b, \pi_\theta] .$$

This quantity measures whether an intervention makes a task or trajectory state more informative for policy optimization, rather than merely increasing the number of generated rollouts.

C.2 Recoverability Regret Analysis

This appendix formalizes the recoverability-regret guarantee for RAIL. Rather than proving learnability of the recoverability function from first principles (), our goal is to show that once recoverability can be tracked with sublinear cumulative error, the resulting intervention controller avoids the linear recoverability gap incurred by fixed or restricted heuristics. The proof uses a standard bandit decomposition: action selection regret can be controlled by the discrepancy between the estimated and true action values, together with an exploration term. This follows the finite-time regret perspective of classical finite-action bandits (Auer et al., 2002a) and the importance-weighted partial-feedback view of adversarial bandits (Auer et al., 2002b). Because RAIL observes side information in the form of trajectory states, we use the oracle-style contextual-bandit perspective, where supervised or cost-sensitive prediction oracles are used to estimate action values under bandit feedback (Langford & Zhang, 2007; Agarwal et al., 2014). Finally, because the recoverability function changes as π_{θ_t} evolves, we model this effect through a variation budget, following classical non-stationary bandit analyses based on switching or drifting reward distributions (Garivier & Moulines, 2011; Besbes et al., 2014; 2015). VIP (Nguyen et al., 2026) provides a closely related variance-informed rollout-allocation theory at the prompt level; RAIL instead studies online regret over a structured state-level intervention space.

Setup. Let $\mathcal{B}$ be a finite intervention space with $K = |\mathcal{B}|$. At training step t, the current policy π_{θ_t} induces a recoverability-gain function

$$f_t(z, b) = \Delta_{\theta_t}(z, b) = V_{\theta_t, b}(z) - V_{\theta_t, \varnothing}(z). \tag{22}$$

Here $z \sim \mathcal{D}_t$ is the rollout state encountered during training and $b \in \mathcal{B}$ is an intervention. The controller selects b_t and observes noisy bandit feedback

$$y_t = f_t(z_t, b_t) + \xi_t, \qquad \mathbb{E}[\xi_t \mid z_t, b_t, \mathcal{H}_{t-1}] = 0, \qquad |\xi_t| \leq \sigma, \tag{23}$$

where $\mathcal{H}_{t-1}$ is the interaction history before step t. The cumulative recoverability regret is

$$\mathrm{Reg}_T = \sum_{t=1}^{T} \mathbb{E}_{z_t \sim \mathcal{D}_t} \left[\max_{b \in \mathcal{B}} f_t(z_t, b) - f_t(z_t, b_t) \right] . \tag{24}$$

This regret compares the selected intervention against the best intervention in the structured intervention space at each step.

Assumptions. We use five assumptions. The first two are standard finite-action contextual-bandit conditions: bounded feedback and sufficient exploration. The next two isolate the main difficulties of our setting: learnability of recoverability and non-stationarity induced by policy optimization. The final assumption is an oracle-style tracking condition that combines these quantities into a cumulative estimation-error bound for the recoverability controller.

Assumption C.3 (Bounded finite-action feedback)**.** *The intervention space $\mathcal{B}$ is finite. and $f_t(z, b) \in [0, 1]$ for all t, z, b. The feedback noise is conditionally mean-zero and bounded as in equation 23.*

Assumption C.4 (Sufficient exploration)**.** *The controller uses a greedy-with-exploration policy: with probability $1 - \rho_t$ it selects the intervention maximizing the current estimate, and with probability ρ_t it explores over $\mathcal{B}$. The exploration schedule satisfies*

$$\Pr(b_t = b \mid z_t, \mathcal{H}_{t-1}) \geq \rho_t / K, \qquad \forall b \in \mathcal{B}, \qquad \sum_{t=1}^{T} \rho_t = \tilde{O}(\sqrt{TK}). \tag{25}$$

This condition ensures nonzero coverage of every intervention. It is the same role played by explicit exploration in finite-action bandits (Auer et al., 2002a) and by importance-weighted exploration in partial-feedback bandit algorithms (Auer et al., 2002b).

How RAIL enforces sufficient exploration. Assumption C.4 is implemented at the level of the finite intervention space, not at the level of primitive language actions. During live training, RAIL uses a greedy-with-exploration policy over the nine executable branch actions in $\mathcal{B}$. With probability $1 - \rho_t$, the controller selects the branch action with the largest predicted recoverability gain if over the live utility gate; with probability ρ_t, it executes an exploratory branch action. This ensures that intervention learning is not driven only by the controller's current argmax, which would otherwise create self-reinforcing blind spots in rarely selected intervention cells.

In implementation, exploration is coverage-directed rather than uniformly random. When the exploration draw fires, RAIL selects the least-covered branch cell among the $|\mathcal{M} \times \mathcal{T}| = 9$ executable interventions, with ties broken deterministically. Thus every intervention cell receives explicit coverage during live training as the controller becomes informative. This exploration probability is decayed over training, so exploration supplies the coverage needed for partial-feedback learning without asymptotically dominating the rollout budget. This realizes the role of ρ_t in Assumption C.4: each branch action receives nonzero exploration support, and the cumulative amount of exploration remains controlled. In this sense, sufficient exploration is guaranteed for learning relative utilities among executable interventions, while the act/skip decision remains governed by the utility gate.

Assumption C.5 (Recoverability learnability)**.** *Let $\mathcal{F}$ be the controller class used to estimate recoverability gains. Define the cumulative best-in-class approximation error*

$$A_T = \sum_{t=1}^{T} \inf_{g \in \mathcal{F}} \mathbb{E}_{z \sim \mathcal{D}_t} \left[\max_{b \in \mathcal{B}} |g(z, b) - f_t(z, b)| \right]. \tag{26}$$

We assume $A_T = o(T)$ in regimes where intervention utility is learnable from outcome traces.

Note this assumption is following the standard practice. We further show that this assumption is also widely supported across tasks via the adaptability experiments we demonstrate in Figure 3.

Assumption C.6 (Controlled policy-induced drift)**.** *The cumulative variation of the recoverability function is bounded by*

$$D_T = \sum_{t=2}^{T} \sup_{z \in \mathcal{Z},\, b \in \mathcal{B}} |f_t(z, b) - f_{t-1}(z, b)|. \tag{27}$$

We assume $D_T = o(T)$ under stable policy optimization.

Assumption C.6 is the stability condition connecting the controller analysis to policy optimization. If policy updates changed recoverability arbitrarily by a constant amount at every step, no online controller could track intervention utility, and the post-training process itself would be unstable. Thus, sublinear drift is a standard tracking premise of policy optimization rather than an additional claim specific to RAIL. Similar variation-budget conditions are commonly used to formalize learnability under changing reward distributions in non-stationary bandits (Besbes et al., 2014; 2015); switching-window analyses provide a related view when the environment changes in segments (Garivier & Moulines, 2011).

This drift perspective also motivates the sliding-window and recency-weighted training. A trace collected at time $j < t$ provides information about f_j, whereas the controller at time t must predict f_t. The mismatch between these two recoverability functions is controlled by the accumulated drift:

$$\sup_{z,b} |f_t(z, b) - f_j(z, b)| \leq \sum_{s=j+1}^{t} \sup_{z,b} |f_s(z, b) - f_{s-1}(z, b)|. \tag{28}$$

Thus, a larger window provides more intervention traces and reduces finite-sample estimation noise, but may introduce stale-data bias when the policy changes. Conversely, a smaller window better tracks the current recoverability function, but relies on fewer samples and can yield noisier estimates. The window size and recency weights therefore balance finite-sample estimation error against policy-induced drift.

Assumption C.7 (Online recoverability tracking)**.** *The online regression procedure used by the recoverability controller produces estimates $\widehat{f}_t$ satisfying the cumulative tracking bound*

$$\epsilon_T := \sum_{t=1}^{T} \mathbb{E}_{z_t}\left[\max_{b\in\mathcal{B}} |\widehat{f}_t(z_t, b) - f_t(z_t, b)|\right] \le \tilde{O}(\sqrt{TK}) + O(D_T) + O(A_T). \tag{29}$$

Assumption C.7 is the main online tracking condition in the theorem. It states that the recoverability controller can follow the evolving intervention-utility function with controlled cumulative prediction error. It compactly captures the contextual-bandit reduction: under sufficient exploration, bandit feedback can be converted into estimable action-value information, and regret can be bounded through the quality of the induced prediction problem. This view is standard in contextual bandits with side information (Langford & Zhang, 2007) and is developed more explicitly through cost-sensitive classification oracle reductions (Agarwal et al., 2014). The variation-budget literature motivates modeling policy-induced non-stationarity through the drift term D_T (Besbes et al., 2014; 2015). In our analysis, the additive contribution of D_T enters through the tracking condition in Assumption C.7. Theorem C.8 then converts this tracking guarantee into an action-selection regret bound.

Main regret theorem. Let $b_t^\star(z) = \arg\max_{b\in\mathcal{B}} f_t(z, b)$ be the best intervention at state z under the current policy. RAIL selects interventions greedily with respect to $\widehat{f}_t$ except for the exploration mass in Assumption C.4.

Theorem C.8 (Recoverability regret under online tracking)**.** *Under Assumptions C.3–C.7, the recoverability controller satisfies*

$$\mathrm{Reg}_T \le 2\epsilon_T + \sum_{t=1}^{T} \rho_t. \tag{30}$$

In particular, if

$$\epsilon_T = \tilde{O}(\sqrt{TK}) + O(D_T) + O(A_T), \qquad \sum_{t=1}^{T} \rho_t = \tilde{O}(\sqrt{TK}), \tag{31}$$

then

$$\mathrm{Reg}_T \le \tilde{O}(\sqrt{TK}) + O(D_T) + O(A_T), \tag{32}$$

where constant factors are absorbed into the $O(\cdot)$ and $\widetilde{O}(\cdot)$ notation. Dividing equation 32 by T gives

$$\frac{\mathrm{Reg}_T}{T} \le \frac{\widetilde{O}(\sqrt{TK})}{T} + \frac{O(D_T)}{T} + \frac{O(A_T)}{T}. \tag{33}$$

Since the intervention space is finite and $K = |\mathcal{B}|$ is constant, the finite-action statistical and exploration term is $o(1)$. Moreover, if $D_T + A_T = o(T)$ with $D_T, A_T \ge 0$, then $D_T/T = o(1)$ and $A_T/T = o(1)$, so the drift and approximation terms also vanish. Therefore,

$$\frac{\mathrm{Reg}_T}{T} \to 0. \tag{34}$$

This shows that the controller may accumulate regret during training, but its average recoverability regret vanishes under sublinear recoverability drift and sublinear cumulative approximation error.

Proof. First consider the greedy part of the controller. Let $b_t = \arg\max_{b\in\mathcal{B}} \widehat{f}_t(z_t, b)$. For any z_t, we have

$$\begin{aligned} f_t(z_t, b_t^\star) - f_t(z_t, b_t) &= f_t(z_t, b_t^\star) - \widehat{f}_t(z_t, b_t^\star) + \widehat{f}_t(z_t, b_t^\star) - \widehat{f}_t(z_t, b_t) \\ &\quad + \widehat{f}_t(z_t, b_t) - f_t(z_t, b_t) \\ &\le 2\max_{b\in\mathcal{B}} |\widehat{f}_t(z_t, b) - f_t(z_t, b)|. \end{aligned} \tag{35}$$

The first term is the estimation error at the truly optimal intervention $b_t^\star$; it can increase the bound when $b_t^\star$ is underestimated. The middle term is non-positive because $b_t \in \arg\max_{b\in\mathcal{B}} \widehat{f}_t(z_t, b)$, so $\widehat{f}_t(z_t, b_t) \geq \widehat{f}_t(z_t, b_t^\star)$. The final term is the estimation error at the selected intervention b_t; it can increase the bound when b_t is overestimated. Thus, regret can arise from underestimating the optimal intervention or overestimating the selected intervention, and both effects are bounded by $\max_{b\in\mathcal{B}} |\widehat{f}_t(z_t, b) - f_t(z_t, b)|$. This gives the factor of two in equation 35.

Exploration contributes at most ρ_t regret at step t because recoverability gains are bounded in $[0, 1]$. Therefore,

$$\begin{aligned}\mathrm{Reg}_T &\leq 2\sum_{t=1}^{T} \mathbb{E}_{z_t}\left[\max_{b\in\mathcal{B}} |\widehat{f}_t(z_t, b) - f_t(z_t, b)|\right] + \sum_{t=1}^{T} \rho_t \\ &= 2\epsilon_T + \sum_{t=1}^{T} \rho_t.\end{aligned} \tag{36}$$

This proves equation 30. Substituting equation 31 gives equation 32. Finally, equation 33 shows that the average regret vanishes when K is finite and $D_T + A_T = o(T)$, proving equation 34. □

Why fixed or restricted intervention rules can incur linear regret. The theorem above establishes that RAIL can track the best structured intervention when recoverability is learnable and policy-induced drift is controlled. We now show why fixed heuristics or restricted action spaces can still incur linear recoverability regret.

Let π^{base} be a baseline intervention policy. If π^{base} is randomized, $f_t(z_t, \pi^{\mathrm{base}}(z_t))$ denotes the expectation over the baseline's action distribution. Define its per-step recoverability gap as

$$\gamma_t(\pi^{\mathrm{base}}) = \mathbb{E}_{z_t}\left[\max_{b\in\mathcal{B}} f_t(z_t, b) - f_t(z_t, \pi^{\mathrm{base}}(z_t))\right]. \tag{37}$$

Proposition C.9 (Persistent mismatch implies linear recoverability regret)**.** *If there exists a constant $\gamma > 0$ and a set of steps $\mathcal{T} \subseteq \{1, \ldots, T\}$ with $|\mathcal{T}| = \Omega(T)$ such that*

$$\gamma_t(\pi^{\mathrm{base}}) \geq \gamma, \qquad \forall t \in \mathcal{T}, \tag{38}$$

then the baseline incurs linear recoverability regret:

$$\mathrm{Reg}_T(\pi^{\mathrm{base}}) = \sum_{t=1}^{T} \gamma_t(\pi^{\mathrm{base}}) \geq \gamma|\mathcal{T}| = \Omega(T). \tag{39}$$

Proof. The result follows by summing over the subset $\mathcal{T}$:

$$\mathrm{Reg}_T(\pi^{\mathrm{base}}) = \sum_{t=1}^{T} \gamma_t(\pi^{\mathrm{base}}) \geq \sum_{t\in\mathcal{T}} \gamma_t(\pi^{\mathrm{base}}) \geq \gamma|\mathcal{T}| = \Omega(T). \tag{40}$$

□

Application to representative baselines. Proposition C.9 is a conditional lower bound: it does not claim that every baseline always fails, but identifies the condition under which a baseline accumulates linear recoverability regret.

- **Fixed uncertainty-triggered branching, e.g., ARPO-style methods.** Such methods use a fixed proxy, such as entropy or uncertainty, to decide when to branch. If this proxy remains persistently misaligned with recoverability. For example, high-uncertainty states are unrecoverable, or low-uncertainty states require recovery intervention. Then the method has a constant per-step recoverability gap on those states. Proposition C.9 then gives linear regret.
- **Prompt-level scalar allocation, e.g., VIP-style methods.** VIP is principled for prompt-level rollout allocation and provides important theoretical motivation for variance-informed budget control (Nguyen et al., 2026). However, relative to RAIL's structured comparator,

VIP selects *how many* rollouts to allocate to a prompt, not *where* or *how* to intervene inside a trajectory. If the best recoverability gain comes from structured choices outside the scalar allocation space, such as branching at a specific trajectory state, changing decoding behavior, or applying a recovery intervention, then scalar allocation has an irreducible representation gap. If this gap persists on a linear number of states, Proposition C.9 implies linear regret with respect to the structured-intervention comparator.

- **Static or proxy-driven intervention policies, e.g., AEPO-style methods.** More generally, any method that maps proxy signals to intervention decisions without updating from realized recoverability outcomes is vulnerable to persistent proxy mismatch. If the proxy-to-gain relation shifts as π_{θ_t} evolves, the method cannot correct its intervention policy from feedback, and its cumulative recoverability regret is linear whenever the mismatch persists.

Takeaway. The regret analysis separates three sources of difficulty: statistical exploration over the finite intervention space, approximation error of the recoverability controller, and policy-induced drift. RAIL achieves sublinear recoverability regret when the structured intervention utility is learnable and changes slowly enough to be tracked. Fixed heuristics and scalar-only allocation methods can incur linear regret whenever their proxy or action-space restriction creates a persistent gap to the best structured intervention. This formalizes the two core gaps in the main text: the non-stationary gap appears through D_T, and the non-scalar gap appears through irreducible representation mismatch.

D Discussions

D.1 Intervention-Shaped Sampling and Policy Updates

RAIL changes the distribution from which training trajectories are collected. Given an intervention b, a continuation is sampled from an intervention-shaped behavior distribution

$$y \sim Q_{\theta_{\text{old}},b}(\cdot \mid z),$$

where $Q_{\theta_{\text{old}},b}$ is induced by the current policy together with the selected branch point, branch budget, and decoding regime, including its temperature and top-p configuration. The policy update still uses the standard GRPO policy ratio between the current policy and the detached pre-update policy. However, it does not use $Q_{\theta_{\text{old}},b}$ as the explicit behavior policy in the denominator. In other words, RAIL does not add an additional importance-correction term for the intervention-shaped decoder beyond the standard GRPO ratio. Instead, the trainer recomputes token log-probabilities under the training policy and applies the standard group-relative update: $\widehat{\mathcal{L}}_{\text{RAIL}}(\theta) = -\frac{1}{|\mathcal{Y}|}\sum_{y_i\in\mathcal{Y}} \widehat{A}_i \sum_\tau \log \pi_\theta(y_{i,\tau} \mid z_i, y_{i,<\tau})$, up to the clipping and masking operations of the underlying GRPO implementation. With the single inner optimization iteration used in our experiments, the reference log-probabilities are detached copies of the current training-policy log-probabilities, so the corresponding policy ratio is one at the update. Thus, branch trajectories generated with different temperatures or top-p values enter the policy loss without temperature reweighting or a denominator based on $Q_{\theta_{\text{old}},b}$.

Accordingly, RAIL should be understood as *intervention-shaped data collection with an approximate on-policy update*. It is on-policy in the limited sense that trajectories are produced by the current policy parameters and are consumed immediately, rather than replayed from an old policy checkpoint. It is not exactly on-policy with respect to the token-level sampling distribution, because the intervention changes that distribution while the loss is evaluated using π_θ. Likewise, our statement that RAIL leaves the optimization objective unchanged refers to the algebraic form of the critic-free policy loss and advantage estimator; RAIL does change the empirical rollout distribution over which that loss is evaluated. One could instead define a corrected update using

$$\frac{\pi_\theta(y \mid z)}{Q_{\theta_{\text{old}},b}(y \mid z)},$$

or its token-level analogue. Such a correction would require tracking the complete intervention-induced behavior probability, including temperature scaling, top-p renormalization, anchor selection, and branching decisions. It would also require suitable support conditions: truncated decoding distributions such as top-p sampling do not generally provide full support for the original policy

distribution. Sequence-level ratios can consequently be unavailable or high-variance, potentially undermining the stable group-relative update that RAIL is designed to preserve.

We therefore adopt the uncorrected formulation deliberately. It isolates RAIL's contribution to the rollout-generation layer, remains compatible with standard critic-free optimization implementations, and avoids introducing a separate high-variance off-policy correction problem. The resulting update should not be interpreted as an unbiased estimator of the original-policy expectation under arbitrary interventions; rather, it optimizes the standard GRPO loss on a selectively reshaped set of trajectories. Developing behavior-corrected intervention learning with explicit support control is a complementary direction beyond the scope of this work.

D.2 Recency-Weighted Tracking under Policy Drift

The recoverability controller learns from historical intervention traces, while its target evolves with the policy. A trace collected at step $j < t$ reflects $f_j(z, b) = \Delta_{\theta_j}(z, b)$, whereas the controller at step t must predict f_t. Retaining more traces reduces statistical estimation error, but older traces may become stale. This is the standard remembering–forgetting trade-off in non-stationary online learning: longer histories improve statistical precision, whereas shorter histories improve responsiveness to change (Besbes et al., 2014). We summarize policy-induced non-stationarity by

$$\delta_t = \sup_{z,b} |f_t(z, b) - f_{t-1}(z, b)|, \qquad D_T = \sum_{t=2}^{T} \delta_t.$$

This variation measure allows recoverability to change smoothly, abruptly, or non-uniformly over training. Its role is to capture the total amount of decision-relevant change, rather than impose a specific temporal model. The additive $O(D_T)$ term in our tracking condition follows from bounded-memory estimation. A trace collected at step j differs from the current target by at most

$$\sup_{z,b} |f_t(z, b) - f_j(z, b)| \leq \sum_{s=j+1}^{t} \delta_s.$$

Under a fixed sliding window or exponential recency weighting with bounded effective memory, each local change δ_s influences only a bounded amount of future training weight. Consequently, the cumulative stale-target error is bounded by a constant multiple of $\sum_s \delta_s = D_T$, and this constant is absorbed into the $O(D_T)$ term. The controller's cumulative tracking error can therefore be interpreted as the sum of statistical estimation error, approximation error, and drift-induced stale supervision.

We emphasize that the condition $D_T = o(T)$ expresses the stability regime required for online tracking. It does not require any particular drift pattern; it only requires that recoverability does not change by a persistent constant amount over a linear number of steps. If instead $D_T = \Omega(T)$, historical outcomes may become obsolete as quickly as they are collected, and vanishing average regret against a dynamic intervention oracle is generally impossible without stronger structure. **Nevertheless, such instability would also undermine the premise that successive policy updates can reuse experience from earlier policies, therefore the discussion of controller's regret becomes meaningless.** Accordingly, $O(D_T)$ should be understood as a sufficient tracking abstraction for RAIL's bounded-memory controller, not as a minimax characterization of arbitrary non-stationary contextual bandits. In practice, RAIL implements this abstraction through exponential recency weighting: recent traces receive greater weight, while older traces decay gradually, balancing statistical stability against adaptation to the evolving policy.

D.3 Entropy-Guided Anchor Selection in Shadow Mode

Entropy is widely used for step-level exploration because it is available before outcome feedback and identifies decisions where the policy assigns substantial probability to competing continuations (Zheng et al., 2025; Wei et al., 2026; Dong et al., 2025a;b). We acknowledge the limitation that the reliance on a shadow warm-up that selects high-entropy anchors may bias the controller's early supervision toward specific uncertainty patterns and could limit coverage of recoverable but low-entropy states. Naturally, low-entropy states may still be recoverable, for example when the policy is confidently wrong in rare cases; however, exhaustive search over all states would be computationally infeasible

and substantially dilute the limited shadow budget. Under a limited shadow-mode budget, high-entropy states provide a practical high-yield proposal set: branching at locally uncertain decisions is more likely to generate diverse continuations than branching indiscriminately across the trajectory. Importantly, RAIL uses entropy only to *propose* anchors, not to determine their recoverability. At each proposed anchor, the iterative procedure evaluates the realized gain of increasing branch budgets and stops once additional branching is no longer useful. Thus, intervention outcomes, rather than entropy itself, determine which trials are retained and provide both positive and negative supervision for the controller. These trajectories are not discarded from policy optimization, but are simply not prioritized for additional branching during controller bootstrapping. The entropy filter therefore represents a deliberate coverage-efficiency trade-off, while iterative outcome-based validation limits dependence on the heuristic proposal signal.

D.4 Design Rationale of the Recoverability Controller

An important design choice in RAIL is to reduce intervention learning from an open-ended generation problem to a lightweight finite-action prediction problem. Directly asking a small controller to generate an appropriate intervention strategy from a language-model trajectory would require modeling a large and poorly structured output space, making lightweight architectures such as MLPs unsuitable. RAIL instead defines a compact set of executable interventions and asks the controller only to estimate the expected recoverability gain of each state–action pair. This reduction substantially simplifies the learning problem while retaining the decisions that matter for rollout intervention: whether, where, and how to branch. **The resulting controller can therefore remain extremely small compared with the policy model and score all candidate interventions at every trajectory state.** In our implementation, this requires only several forward passes through a lightweight model, whose cost is negligible compared with generating even a single additional LLM continuation. This computational asymmetry is important: because intervention scoring is essentially free relative to rollout generation, RAIL can repeatedly evaluate intervention utility throughout a trajectory and use the utility gate to decide dynamically both where branching is worthwhile and which branching strategy should be applied. **It also motivates our deliberate choice of an MLP rather than a Transformer or another heavier controller.** Once the intervention space has been reduced to compact state–action utility estimation, additional controller capacity would introduce unnecessary inference and optimization cost without addressing a correspondingly complex prediction problem. Thus, the lightweight controller is not merely an implementation convenience, but a consequence of how RAIL structures the intervention-learning problem.

D.5 Recoverability Beyond Binary Rewards

RAIL instantiates recoverability through intervention-induced reward contrast rather than treating reward variance as a surrogate for final task success. As analyzed in Section 2.1 and Appendix C.1, under the binary verifier rewards used in all of our experiments, this choice has a particularly clean interpretation: reward variance is exactly $p(1-p)$, vanishes for all-success or all-failure rollout groups, and is maximized at $p = 0.5$, while our analysis further connects it directly to the magnitude of relative-advantage and projected-gradient signals. Thus, in our setting, increasing recoverability means exposing additional learning signal for group-based optimization, whose downstream value is ultimately evaluated by final task success. However, this interpretation becomes less direct with dense or partial rewards. Although reward variance continues to measure relative-reward contrast, larger variance need not indicate that the newly exposed trajectories are more recoverable in the desired direction; dispersion can increase without a corresponding improvement in expected reward. Extending RAIL to such settings may therefore require a richer intervention objective that jointly accounts for reward contrast and reward improvement, or more directly estimates the utility of an intervention for the subsequent policy update. We leave this generalization to broader reward structures for future work.

E Prompt Design

In this section, we provide the prompts for all four agentic benchmarks. We utilize a Reasoning-Then-Act template for the training. These components are essential for interpretability and reproducibility.

AgentBench OS Prompt

Role and Goal

```
You are an AI assistant with full access to a Linux (Ubuntu)
operating system through bash commands. You can execute bash commands
to gather information, navigate directories, read files, and
manipulate data. Your goal is to solve the given task using the
provided tools.
```

Available Tools

```
- bash_action: Execute bash commands in the Linux environment. Use
this tool to gather information and perform the requested operations.
- answer_action: Submit the final answer when the result is known.
```

Interaction Protocol

```
Before calling a tool, briefly state your reasoning in one sentence
explaining what you plan to do and why. Then immediately call exactly
ONE tool using <tool_call> tags.

Every turn must contain exactly one tool call. Use bash_action for
commands such as grep, find, cat, awk, wc, and ls. Do not use
interactive commands or commands requiring user input. When the
answer is known, call answer_action with only the exact result value,
such as "42" rather than "the answer is 42". If a command fails, use
the error to choose a different approach.
```

Examples

```
User: How many .txt files are in /home?
Assistant: I need to find and count all .txt files in /home using
find and wc.
<tool_call>
{"name":"bash_action","arguments":{"script":"find /home -name '*.txt'
-type f | wc -l"}}
</tool_call>

Tool: 5
Assistant: The command returned 5, which is the answer.
<tool_call>
{"name":"answer_action","arguments":{"answer":"5"}}
</tool_call>

Always provide brief reasoning before calling a tool.
```

Figure 8: System prompt used for **AgentBench OS**. Section labels are added only for presentation and organize the role, available tools, interaction protocol, and demonstrations.

AgentBench DB Prompt

Role and Goal

```
You are an AI assistant with access to a MySQL database. You can
execute SQL queries to retrieve data or modify the database. Your
goal is to solve the given task using the provided tools.
```

Available Tools

```
- sql_query: Execute SELECT, INSERT, UPDATE, or DELETE queries.
- answer_action: Submit the final answer.
```

Interaction Protocol

```
Before calling a tool, briefly state your reasoning in one sentence
explaining what you plan to do and why. Then immediately call exactly
ONE tool using <tool_call> tags.

Always use backticks around table and column names. For SELECT tasks,
call answer_action with only the query result value, such as "John
Smith", "42", or "26.0". For INSERT, UPDATE, or DELETE tasks, first
execute the write query using sql_query and then call answer_action
with a short confirmation such as "done"; the grader evaluates the
resulting database state rather than the answer text.

If a query fails, use the error to construct a corrected query and do
not repeat the same query. If a successful query returns an empty
result, verify the schema, table and column names, filter value, case
sensitivity, exact-string matching, and data types. In particular,
numerical comparisons over text columns may require CAST. Return "[]"
or "No results" only when you are confident that the result is
genuinely empty.
```

Question Interpretation

```
Some questions use automatically generated phrasing. "Name the X",
"Tell me X", "Name the most X", and "Name the least X" ask for the
VALUE of column X, which may be a number, date, or string, rather
than a person's name. For example, "Name the most age" asks for the
maximum age value. "How many X" expects only a count value.
```

Examples

```
User: What is the name of the person with ID 42 from the "User Data"
table?
Assistant: I will retrieve the name associated with ID 42.
<tool_call>
{"name":"sql_query","arguments":{"query":"SELECT `name` FROM `User
Data` WHERE `id` = 42"}}
</tool_call>

Tool: John Smith
Assistant: The query returned John Smith, so that is the answer.
<tool_call>
{"name":"answer_action","arguments":{"answer":"John Smith"}}
</tool_call>

Begin by understanding the task.
```

Figure 9: System prompt used for **AgentBench DB**. Section labels are added only for presentation and organize the database interaction rules, answer protocol, question interpretation, and demonstrations.

WebShop Prompt

Role and Goal

```
You are a shopping agent in a text-based online store. Your task is
to purchase the ONE product that best matches the user's instruction,
including the requested product type, attributes, options, and price.
```

Available Tools

```
- search_action: Search the store using keywords when a search bar is
available.
- click_action: Click an exact label shown in the current observation,
including a product ID, option value, navigation control, information
tab, or "Buy Now".
- answer_action: End the session when purchasing cannot be completed.
Normally, prefer completing the task through "Buy Now".
```

Store Interaction

```
The current observation lists all clickable elements. Begin by
searching with focused keywords derived from the instruction. Search
results contain product IDs, titles, and prices. Click a promising
product ID to open its page, inspect its description or features when
necessary, select every requested option such as color or size, and
click "Buy Now" to complete the purchase.

You are evaluated by how well the purchased product satisfies all
requested attributes and options while remaining within the price
limit. Refine the search when the results are poor, and click only
labels that appear in the current observation.
```

Interaction Protocol

```
Every turn must contain one brief sentence of reasoning followed
immediately by exactly ONE tool call using <tool_call> tags.
```

Example

```
Assistant: I will search for the requested product using its main
attributes.
<tool_call>
{"name":"search_action","arguments":{"query":"machine washable
decorative pillows 28 inch"}}
</tool_call>

Tool: [Results] B07XYZ1234  Machine Washable Decorative Throw
Pillows.
Assistant: B07XYZ1234 matches the request, so I will open it.
<tool_call>
{"name":"click_action","arguments":{"button":"B07XYZ1234"}}
</tool_call>

Tool: [Product] options: color [blue] [gray]; size [28" x 28"] ...
[Buy Now]
Assistant: I will select the requested size before purchasing.
<tool_call>
{"name":"click_action","arguments":{"button":"28\" x 28\""}}
</tool_call>

Tool: [Product] selected size: 28" x 28" ... [Buy Now]
Assistant: The product and selected options match the instruction, so
I will complete the purchase.
<tool_call>
{"name":"click_action","arguments":{"button":"Buy Now"}}
</tool_call>


Begin.
```

Figure 10: System prompt used for **WebShop**. Section labels are added only for presentation and organize the shopping objective, available actions, environment workflow, and tool-calling protocol.

ToolQA-Coffee Prompt

Role and Data

```
You answer questions about coffee prices using a Python interpreter.
A pandas DataFrame named df is preloaded with the columns Date, Open,
High, Low, Close, and Volume. Date is a string in YYYY-MM-DD format;
Open, High, Low, and Close are floating-point values; and Volume is
an integer. You must query df to obtain the requested values rather
than answering from memory.
```

Available Tools

```
- python_interpreter: Execute Python code with df and pandas
available. Assign the computed result to a variable named ans.
- answer_action: Submit only the final answer value.
```

Interaction Protocol

```
On each turn, provide one sentence of reasoning followed immediately
by exactly ONE tool call using <tool_call> tags. Use
python_interpreter to query or compute over df and assign the result
to ans. After observing the result, call answer_action with only the
value, such as a number, "bullish", or "bearish".
```

Tool-Call Formatting

```
The <tool_call> body must be valid JSON. The value of "code" must be
one double-quoted string. Use single quotes inside the Python code,
encode internal line breaks as \n, and always close the code string
with a final double quote before the closing braces.
```

Example

```
User: Was 2015-07-29 a bullish or bearish day for coffee?
Assistant: I will compare the closing and opening prices for that
date.
<tool_call>
{"name":"python_interpreter","arguments":{"code":"row = df[df.Date ==
'2015-07-29']\nans = 'bullish' if row.Close.iloc[0] >
row.Open.iloc[0] else 'bearish'\nprint(ans)"}}
</tool_call>

Tool: bearish
Assistant: The computed result is bearish.
<tool_call>
{"name":"answer_action","arguments":{"answer":"bearish"}}
</tool_call>

Begin.
```

Figure 11: System prompt used for the **ToolQA-Coffee** benchmark. Section labels are added only for presentation and organize the preloaded data interface, available tools, execution protocol, JSON constraint, and demonstration.