RECOVER OR CASCADE? THE HIDDEN ROLE OF KL REGULARIZATION IN MULTI-TURN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Preventing errors from propagating through a multi-turn language-model agent is important for reliable task execution. However, evaluations of agent training commonly report accuracy and final task success, which do not distinguish avoiding errors from recovering after them. We study how on-policy reinforcement learning (RL) shapes behavior after errors by varying the coefficient β in penalties motivated by Kullback–Leibler (KL) regularization. The intended constraint on departures from a reference policy provides a way to examine whether training preserves existing behavior or allows corrective actions to be learned. We measure failure persistence, the probability of failing after a failed step, and sequential consistency, the probability of passing after a passed step. Across five main benchmark settings, failure persistence is higher at $\beta = 1$ than at $\beta = 0$ in four. Our analyses further reveal that, as β changes, higher sequential consistency can coexist with a lower probability of returning to a passing step after failure, distinguishing gains in continued correctness from gains in recovery.

1 INTRODUCTION

Language model agents solve tasks through sequences of actions, such as searching for information, calling tools, and executing code (Qi et al., 2025; Seed, 2025; Patil et al., 2024; Jimenez et al., 2024). An incorrect action can change the next observation and make later decisions harder. The agent may use feedback to correct its mistake and complete the task, or enter an *error cascade*, where the initial mistake leads to further mistakes. We therefore ask: *what makes errors persist in multi-turn agents, and how do training choices affect their ability to recover?*

Prior work explains error accumulation outside a policy's demonstration distribution (Ross et al., 2011) and examines the difficulty of combining decisions into longer sequences (Pan et al., 2024). Work on agent training and feedback-based correction studies how to improve task outcomes (Qi et al., 2025; Xi et al., 2025; Madaan et al., 2023; Shinn et al., 2023). RAGEN-2 examines *template collapse*, where outputs remain varied but depend less on the input (Wang et al., 2026).

However, how training constraints change the agent's next decision after an error remains unclear. As a key approach to agent post-training, on-policy reinforcement learning (RL) updates a policy using rewards from its own generated sequences of actions and observations, called trajectories (Qi et al., 2025; Xi et al., 2025). These trajectories include the agent's errors and subsequent actions, so reward-based updates can reinforce or discourage the behavior that follows a mistake. We therefore study on-policy RL to examine how training shapes decisions after errors. Within this training process, Kullback–Leibler (KL) regularization provides our entry point: its intended role is to penalize departures from a reference policy (Shao et al., 2024). Learning to recover may require increasing the probability of corrective actions that the reference policy rarely takes. A stronger reference-policy constraint could resist that change while retaining useful prior behavior, making its coefficient β a relevant variable for studying post-error learning. We compare behavior across β and examine the implemented loss to determine the update that the coefficient actually controls.

To examine error propagation and recovery from both behavioral and optimization perspectives, we measure post-error decisions and the updates that shape them. We define *failure persistence*, $P(f|f)$, and *sequential consistency*, $P(p|p)$, which measure the probabilities of failing after a failed step and passing after a passed step, respectively. The complementary rate $1 - P(f|f)$ measures

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.

observed recovery, a return to a pass under the benchmark's step-labeling rule. We compare these rates across coefficients, then shuffle step labels within each trajectory while preserving its failure count to separate information about failure order from overall failure frequency. We further analyze how β changes reward-based updates, connecting these behavioral comparisons to the implemented training loss.

Our five main benchmark settings cover tool use, question answering, and interactive tasks. The main comparisons show higher failure persistence at $\beta = 1$ than at $\beta = 0$ in four of five settings, with the largest increase in the WebShop setting, from 0.101 to 0.801. The coefficient with the lowest failure persistence varies by task, and supplementary proximal policy optimization (PPO) comparisons show that coefficient rankings can also differ across training algorithms. To explain these behavioral differences, we combine loss analysis and direct gradient measurements with observation-replacement tests and tool-call error analysis, examining both how training changes the policy and how the resulting agent responds during a task. Our analyses further reveal that β affects continued correctness and observed recovery differently: improved sequential consistency can coexist with weaker recovery after errors.

Overall, our contributions are as follows:

1. We study error propagation and recovery through training constraints, using the KL coefficient to connect policy updates with post-error decisions.
2. We combine failure persistence, sequential consistency, and label shuffling to distinguish failure order from frequency, and use gradients and observation swaps to examine updates and feedback use.
3. Across five main settings, four show higher failure persistence at $\beta = 1$ than at $\beta = 0$, while the coefficient with the lowest failure persistence varies by task. These findings motivate evaluating training constraints for multi-turn agents through observed recovery as well as overall task performance.

2 Related Work

Error propagation and recovery. Error-propagation studies examine how a policy's actions affect later states. Imitation-learning theory shows that errors can accumulate when a learned policy visits states that differ from those in its demonstrations (Ross et al., 2011). Task-composition benchmarks show that performance can drop when individually solvable subtasks must be completed in a longer sequence (Pan et al., 2024). Other work trains agents to use feedback for correction: reinforcement learning updates policies from task feedback (Qi et al., 2025; Xi et al., 2025), while Self-Refine and Reflexion revise outputs or store feedback for later attempts (Madaan et al., 2023; Shinn et al., 2023). RAGEN-2 studies a related failure mode, *template collapse*, in which outputs become less dependent on the input even when they remain diverse (Wang et al., 2026). These studies connect error propagation, feedback, and input use. They usually evaluate overall task success, later attempts, or input dependence, rather than the next decision after a failed step within the same trajectory. We build on them by measuring failure persistence and sequential consistency within trajectories, then testing whether failure order adds information beyond frequency.

The role of KL regularization. Both Proximal Policy Optimization (PPO)-based language-model training and the original Group Relative Policy Optimization (GRPO) formulation use KL penalties to constrain departures from a reference policy (Shao et al., 2024). This reference-policy constraint differs from PPO's clipping, which limits updates relative to the policy used to collect samples (Schulman et al., 2017). PPO-based reinforcement learning from human feedback (RLHF) commonly adds the KL penalty to rewards, while the original GRPO objective includes it directly as a regularization term (Shao et al., 2024). GRPO estimates advantages by comparing rewards within groups of sampled outputs, avoiding a separate value model that predicts expected return (Shao et al., 2024). The lower training cost motivates our main GRPO comparisons; PPO tests whether the findings change with the algorithm. Related studies examine reward-model overoptimization (Gao et al., 2023) and how RLHF affects generalization and output diversity (Kirk et al., 2024). These studies motivate balancing reward-driven changes against prior behavior. For multi-turn agents, we ask whether this constraint changes the decision after failure feedback. We use the KL coefficient to study this question under

GRPO and PPO. Our focus is behavior after failure within the same trajectory, linking coefficient comparisons to failure persistence, feedback use, and the implemented update.

3 Analyzing Error Persistence and Recovery

To study how β relates to error propagation and recovery, Section 3.1 defines behavior after failed and passed steps. Section 3.2 specifies coefficient comparisons and controls for failure frequency; Section 3.3 defines loss and gradient analyses to examine how β changes reinforcement. Section 4 specifies the experimental settings for these analyses, and Section 5 reports their results and interpretation.

3.1 Failure Persistence and Sequential Consistency

An *episode* is one task trajectory, with each scored step labeled pass (p) or fail (f) under the benchmark's rule.[1] *Failure persistence*, $P(f \mid f)$, is the probability of failing immediately after a failed step; *sequential consistency*, $P(p \mid p)$, is the probability of passing immediately after a passed step. We call $P(p \mid f) = 1 - P(f \mid f)$ *observed recovery*: returning to a passing step under the scoring rule, distinct from completing the task. We pool adjacent-step counts within each run; hats denote estimates, such as $\widehat{P}(f \mid f)$. Pairs never cross episode boundaries; if no pair starts with a failure or a pass, its corresponding rate is undefined, not zero.

For example, the illustrative sequences (p, f, p, f, p, f) and (p, p, p, f, f, f) both pass three of six steps, but their failure persistence is $0/2$ and $2/2$, respectively. The first returns to a pass after both failures that have a following step; the second never does, despite having the same overall accuracy.

3.2 Measuring Changes in Error Persistence and Recovery

Measuring post-error behavior across coefficients. We compute failure persistence and sequential consistency at each tested β and compare coefficients separately for each model and benchmark. We report the full curves and the endpoint change $\widehat{P}(f|f)_{\beta=1} - \widehat{P}(f|f)_{\beta=0}$; a positive change means greater persistence at the high endpoint, not necessarily a monotonic trend.

Comparing transition rates with accuracy and outcomes. Select an adjacent-step pair uniformly from a run; p_{t-1} and p_t denote passes at its two positions, and f_{t-1} a failure at the first. When both starting labels occur, the next-step pass probability combines sequential consistency and observed recovery:

$$P(p_t) = P(p_t \mid p_{t-1})P(p_{t-1}) + P(p_t \mid f_{t-1})P(f_{t-1}). \tag{1}$$

The left side excludes initial steps; the right side weights consistency and recovery by their starting-label frequencies. Reporting both rates alongside accuracy and task outcomes distinguishes maintaining correct behavior from returning to it.

Testing failure order beyond failure frequency. A high $P(f \mid f)$ may simply reflect frequent failure, so we repeatedly shuffle labels within each episode while preserving its length and pass/fail counts. Comparing observed rates with the shuffled mean and variation tests whether order adds information beyond frequency.[2] Only labels are reordered: the actions and recorded task outcomes remain those of the original episodes.

3.3 Training-Gradient Analysis

To relate behavioral differences to training, we examine how β changes the reinforcement of rewarded actions.

[1] Appendix A.3 gives each benchmark's rules and labeled examples.

[2] Appendix B.2 details shuffling and statistical comparisons for evaluation on the same 100 held-out HotPotQA tasks.

Measuring the effect of β on the implemented loss. KL regularization is intended to penalize departures from a reference policy; interpreting β requires examining the implemented loss. For HotPotQA, InterCode, ALFWorld, and WebShop, $\bar{\ell}_\theta$ and $\bar{\ell}_{\text{ref}}$ are mean log probabilities over the same sampled assistant tokens, with trainable parameters θ and a fixed reference model. Let A be the episode's normalized reward advantage, positive when its reward exceeds the mean for episodes sampled on the same task. Holding A and sampled tokens fixed, the loss and its gradient are

$$L = -A\bar{\ell}_\theta + \beta(\bar{\ell}_\theta - \bar{\ell}_{\text{ref}}), \qquad \nabla_\theta L = (\beta - A)\nabla_\theta \bar{\ell}_\theta. \tag{2}$$

The *effective advantage* is $A - \beta$: if $\beta > A > 0$, the loss discourages a rewarded response before averaging and optimizer transformations. The reference log probabilities are constant during differentiation, so this fixed-sample gradient does not depend on the reference distribution.[3]

Measuring penalty and task-gradient contributions. Let $\bar{g}_{\text{pg}}$ and $\bar{g}_{\text{KL}}$ average the gradients of $-A\bar{\ell}_\theta$ and $\bar{\ell}_\theta - \bar{\ell}_{\text{ref}}$ across episodes; the subscript KL denotes the implemented penalty. The *penalty-gradient ratio* compares their magnitudes, with $\|\cdot\|$ denoting the Euclidean norm:

$$\rho(\beta) = \frac{\beta\|\bar{g}_{\text{KL}}\|}{\|\bar{g}_{\text{pg}}\|}, \qquad \|\bar{g}_{\text{pg}}\| > 0. \tag{3}$$

Here $\rho > 1$ means the weighted penalty gradient is larger than the task gradient. Their *gradient cosine* divides their inner product by the product of their nonzero norms; a negative value indicates opposing components. Task-gradient signal-to-noise ratio (SNR) is the squared norm of the mean task gradient divided by the mean squared deviation of episode gradients from that mean. It measures task-gradient agreement across episodes, separately from penalty strength. We compare gradient magnitude, direction, and agreement across β, then relate these measurements to failure persistence under the sampling and averaging protocol in Section 4.3.

4 Experimental Setup

4.1 Models and Training

The main experiments use Qwen3 (Team, 2025) with GRPO: 1.7B on BFCL and 4B on HotPotQA, InterCode-SQL (InterCode), ALFWorld, and WebShop. We use "rollout" for one task trajectory sampled by the current policy. We compare $\beta \in \{0, 0.1, 0.5, 1.0\}$ on BFCL and additionally include $\beta = 0.01$ in the four 4B settings. The main runs use AdamW with learning rate 10^{-5}; Equation 2 specifies the sampled loss in the four-benchmark trainer. PPO comparisons use Qwen3-4B at $\beta \in \{0, 0.1, 0.5, 1\}$ on HotPotQA, WebShop, and InterCode, reporting final training-batch rates; Appendix C.3.4 details their budgets and clipping settings. Supplementary comparisons cover PPO, 0.6B and 8B models, Llama-3.1 (Team, 2024), and τ-bench.[4]

4.2 Benchmarks

The five main settings are BFCL v4 (Patil et al., 2025) for function calls, HotPotQA (Yang et al., 2018) for retrieval-based question answering, InterCode (Yang et al., 2023) for SQL, ALFWorld (Shridhar et al., 2021) for household actions, and WebShop (Yao et al., 2022) for shopping. Our ALFWorld and WebShop settings replay fixed observations and score reference-action matches; HotPotQA selects supplied context paragraphs, while InterCode uses execution results or SQL-token matching. BFCL compares functions and arguments with references; the additional long-horizon τ-bench retail setting (Yao et al., 2025) aligns executed application programming interface (API) calls with reference sequences.[5]

[3] Appendix D.1 gives the derivation, numerical check, and implementation scope; Appendix A.1 specifies reward-advantage normalization.

[4] Appendix A.1 details the main training configurations, including optimization and generation settings. Appendix C.3 details the 8B, turn-budget, PPO, and Llama comparisons; Appendix C.1.4 describes the additional InterCode and ALFWorld runs.

[5] Appendix A.3 gives scoring rules and labeled examples for all six settings.

4.3 Evaluation Protocols

Sampling and checkpoints. Each coefficient setting uses the same model size and benchmark-specific step-labeling rule within a benchmark. The four 4B benchmarks use the last training batch sampled before the final update, while BFCL evaluates the trained 1.7B models separately; Appendix A.2 details budgets and timing.

Held-out HotPotQA evaluation. The Qwen3-4B base model and one final checkpoint per coefficient are evaluated on the same 100 held-out HotPotQA tasks. Each episode records step labels and a final-answer score. We shuffle labels within each episode 5,000 times, preserving its length and counts, and apply one-sided tests with Holm correction across six conditions and two rates. The evaluator requests at most eight steps, temperature 0.01, and 256 generated tokens per step.

Training-gradient measurements. Gradient measurements use separate Qwen3-4B GRPO runs on HotPotQA at $\beta \in \{0, 0.1, 0.5, 1\}$. Before updates, we measure both gradients on nonzero-advantage trajectories in the final decoder block, output head, and final layer norm. We average valid gradient diagnostics across iterations and report final-batch failure persistence; Appendix D.2 details sampling and averaging.

Computing failure persistence and sequential consistency. Following Section 3.1, we pool within-episode transition counts per run, divide by the corresponding starting-label counts, and exclude empty denominators. We summarize training runs by their mean and one standard error; held-out rate intervals use 5,000 whole-episode bootstrap resamples, detailed in Appendix B.2. Per-turn accuracy is the passing fraction at a given turn; held-out HotPotQA reports the fraction over all scored steps and the number of answers with overlap at least 0.5.[6]

5 Results

5.1 Error Propagation and Recovery across β

Table 1: Failure-persistence ranges and endpoint changes across β for the five main benchmark settings. The endpoint change is $\widehat{P}(f|f)_{\beta=1} - \widehat{P}(f|f)_{\beta=0}$.

Benchmark	Range of $P(f\|f)$	Max. − min.	Endpoint change
BFCL	.833–.994	.161	+.161
HotPotQA	.764–.860	.096	−.023
InterCode	.761–.919	.158	+.087
ALFWorld	.728–.866	.137	+.081
WebShop	.101–.801	.700	+.700

A larger β can accompany greater failure persistence. At $\beta = 1$, failed steps are less often followed by a pass than at $\beta = 0$ in four of the five main settings. Mean $P(f|f)$ rises from 0.101 to 0.801 on WebShop and from 0.833 to 0.994 on BFCL, with endpoint increases of 0.087 on InterCode and 0.081 on ALFWorld (Figure 1; Table 1). In WebShop, observed recovery falls from 89.9% to 19.9% under the same scoring rule. In the four-benchmark loss, increasing β weakens reinforcement of above-average-reward trajectories, including any corrective actions they contain.

The β with the lowest failure persistence depends on the task. BFCL has its lowest $P(f|f)$ at $\beta = 0$, whereas HotPotQA has its lowest mean at $\beta = 0.1$ and a lower mean at $\beta = 1$ than at $\beta = 0$ by 0.023.[7] The step labels test different behavior: BFCL requires an exact function-and-argument match, whereas HotPotQA scores retrieval and final answers through answer-token overlap. These rules determine which actions count as a return to a pass, so the best coefficient must be assessed against each task's feedback and scoring rule.

[6] Appendix A.3 defines answer overlap; Appendix A.4 checks how labels relate to task progress and rewards.

[7] Appendix C.2.1 details BFCL rates across coefficients; Appendix C.1.2 reports the HotPotQA runs underlying Figure 1.

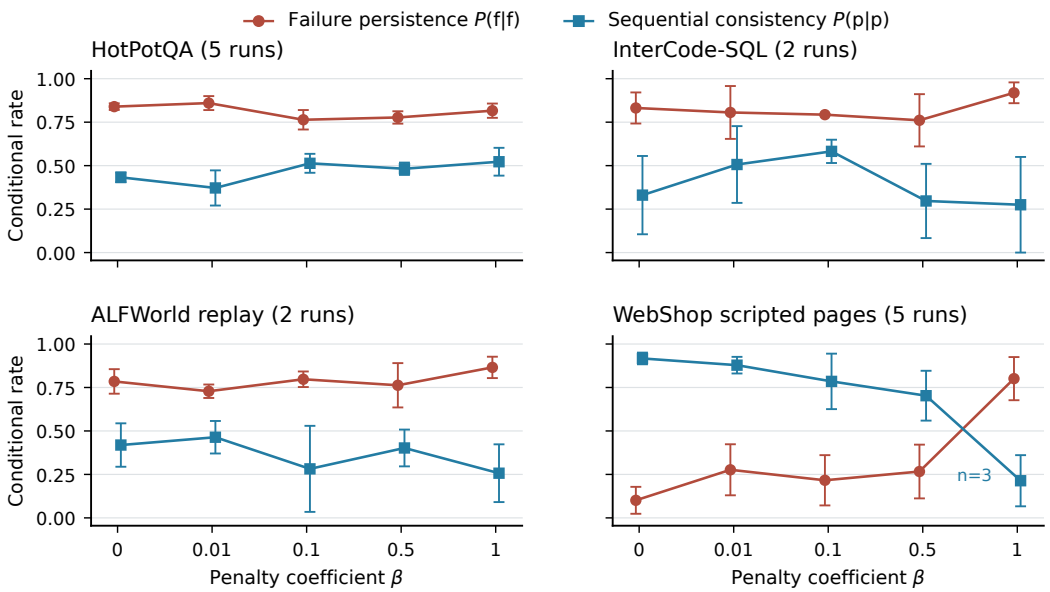

Figure 1: Failure persistence $P(\text{f}|\text{f})$ and sequential consistency $P(\text{p}|\text{p})$ across β on four Qwen3-4B benchmark settings. Curves summarize final training batches with one-standard-error bars across runs; n counts runs with a defined rate.

The effect of increasing β on failure persistence depends on the task and algorithm. Under PPO, $\beta = 1$ gives the lowest measured $P(f|f)$ on HotPotQA (0.841), but the highest or tied highest on WebShop and InterCode (1.000 on both).[8] In the HotPotQA comparison with the same task and iteration counts, increasing β from 0.5 to 1 lowers $P(f|f)$ under PPO (0.852 to 0.841) but raises it under GRPO (0.817 to 0.898). GRPO and PPO produce different reward-based updates that interact with the penalty, motivating task- and algorithm-specific selection of β.

5.2 Transition Rates Distinguish Sustained Correctness from Observed Recovery

Sequential consistency and observed recovery respond differently to β. In the BFCL evaluation, increasing β from 0 to 0.1 raises the probability of passing after a passed step from 4.2% to 13.0%. Over the same comparison, observed recovery is 16.7% at $\beta = 0$ and 15.9% at $\beta = 0.1$.[9] Equation 1 shows why these behaviors must be measured separately: next-step accuracy averages the two probabilities, weighted by how often the previous step passes or fails. Reporting the two rates separately locates the measured improvement in continuing correctly after a pass and directly compares it with behavior after failure.

Low pass rates account for much of the high failure persistence. In the evaluation on the same 100 held-out HotPotQA tasks, trained policies have $P(f|f) = 0.956$–0.988, compared with shuffled means of 0.949–0.983 (Figure 2). For the individual trained policies, shuffling lowers failure persistence by 0.53–1.34 percentage points. This control preserves episode lengths and pass/fail counts, changing only label order (Section 3.2). Only 2.0–9.4% of scored steps pass (Table 2), so even a random ordering leaves few passing steps to interrupt failures. The shuffled baseline thus quantifies the persistence expected from each episode's pass/fail counts, while the observed excess measures the additional contribution of their order.

Failure order reveals whether passing steps interrupt failures. At $\beta = 1$, passing steps follow failures less often than the shuffled baseline predicts, despite identical pass counts and recorded

[8] Appendix C.3.4 gives the PPO coefficient comparisons and training budgets for all three benchmarks.
[9] Appendix C.2.1 reports both probabilities across the tested BFCL coefficients.

Table 2: Held-out HotPotQA transition and outcome statistics on the same 100 tasks for each condition. Pass is the fraction of passing scored steps, Answer is the number of episodes with answer overlap at least 0.5, and brackets give 95% episode-bootstrap confidence intervals (CIs).

Condition	Pass	Answer	P(f|f) [95% CI]	P(p|p) [95% CI]
Base	.174	28/100	.869 [.827, .904]	.446 [.317, .562]
$\beta = 0$	.023	1/100	.986 [.977, .994]	.188 [.000, .429]
$\beta = .01$	.030	1/100	.988 [.979, .995]	.500 [.083, .771]
$\beta = .1$	.059	5/100	.956 [.935, .973]	.342 [.212, .452]
$\beta = .5$	.020	0/100	.988 [.976, .997]	.143 [.000, .278]
$\beta = 1$	.094	5/100	.962 [.943, .979]	.468 [.200, .646]

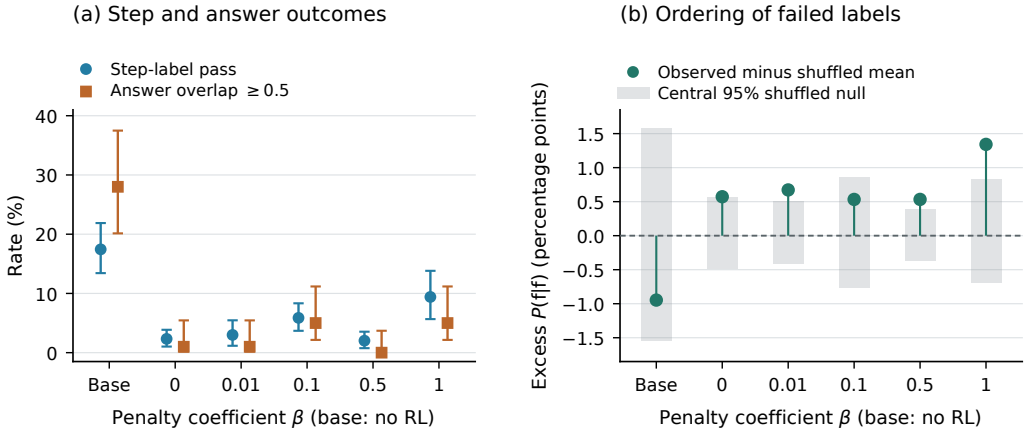

Figure 2: Outcomes and failure-order diagnostics from evaluation on the same 100 held-out HotPotQA tasks. Left: step-label and answer-overlap rates; right: observed minus shuffled failure persistence.

answer outcomes. Observed recovery is 3.8%, compared with 5.1% after shuffling; equivalently, failure persistence exceeds its baseline after Holm correction for 12 comparisons ($p = .0048$). Sequential consistency also exceeds its baseline for the base model and $\beta = .1$ ($p = .0400$ and .0110), indicating that passes can cluster as well.[10] Accuracy counts how many steps pass, whereas transition rates locate those passes relative to earlier failures: they distinguish sustained correct behavior from a return to a pass after an error.

5.3 Effect of the Training Penalty on Policy Updates

A larger β weakens reinforcement of positive-advantage trajectories. In the implemented four-benchmark GRPO loss, β sets the reward-advantage threshold above which a sampled trajectory receives positive reinforcement. Equation 2 gives effective advantage $A - \beta$: at fixed samples, reinforcement vanishes at $\beta = A$ and reverses when $\beta > A > 0$. The penalty subtracts from each trajectory's advantage regardless of whether it contains a useful correction. A corrective action can therefore occur in a rewarded trajectory yet receive no positive reinforcement from that sample, linking the loss directly to the learning of post-error behavior.[11]

The penalty changes the direction of the combined loss gradient. On the measured output-related parameters in the separate HotPotQA experiment, the iteration-averaged penalty-gradient ratio is 2.64 and the mean cosine is -0.145 at $\beta = 1$ (Table 3). The penalty gradient is larger than the task gradient in both valid measurement iterations, but their cosine changes from positive to negative. When the two gradients differ in direction, increasing β changes their combined direction rather than merely scaling the task gradient. Measuring magnitude and cosine together reveals both the

[10] Appendix B.2 reports the label-shuffling procedure, intervals, and corrected tests for all six conditions.
[11] Appendix D.1 derives this fixed-sample gradient and specifies the implementations it covers.

Table 3: Gradient diagnostics and failure persistence during Qwen3-4B GRPO training on HotPotQA. Gradient columns average valid iterations; $P(\mathrm{f}|\mathrm{f})$ comes from the final training batch of these separate runs.

β	ρ	Grad SNR	Cosine	$P(\mathrm{f}\|\mathrm{f})$
0.0	0.000	0.106	0.044	0.944
0.1	0.353	0.076	0.205	0.940
0.5	1.526	0.109	−0.022	0.846
1.0	2.643	0.172	−0.145	0.880

penalty's relative strength and whether its component along the task gradient reinforces or opposes that gradient.

A stronger penalty contribution can coexist with lower failure persistence. From $\beta = 0.1$ to 0.5, mean ρ rises from 0.353 to 1.526, yet mean task-gradient SNR rises from 0.076 to 0.109 and final-batch $P(f|f)$ falls from 0.940 to 0.846. Unlike a fixed-sample comparison, training at different β changes the policy, sampled trajectories, and task gradients. The penalty can therefore become stronger relative to the mean task gradient while the task gradients become more consistent across trajectories.

6 Analysis

To understand the behavioral differences in Section 5, we first test responses to changed observations, since feedback can guide correction (Section 6.1). We then examine which tool-call errors accompany persistent failures (Section 6.2)[12], before asking whether reasoning before acting improves performance at different coefficients (Section 6.3).

6.1 Does training change the agent's response to new observations?

Across all tested training coefficients, trained agents change their responses less often than the base model when the latest observation is replaced. We test base and trained policies on HotPotQA by replacing the latest observation with one from another task, retaining the rest of each history. We define the response-text change rate as the fraction of replacements that change the response under a word-overlap rule, averaged within each turn and then across turns.[13] The rate is 0.716 for the base model but only 0.067–0.263 for trained policies (Table 4), showing lower sensitivity to new observations in every trained condition. This reduced sensitivity identifies a potential obstacle to feedback-based correction: new information may leave the agent's response unchanged even when revising its decision would help.

Table 4: Observation-swap response rates on HotPotQA for Qwen3-4B policies. The change rate averages response changes across three replacements per turn; N is the number of tested turns.

Condition	Response-text change rate	Tested turns N
Base (no RL)	.716	81
$\beta = 0.0$	.114	99
$\beta = 0.1$	.195	94
$\beta = 0.5$	.067	105
$\beta = 1.0$	.263	105

6.2 Which action errors accompany persistent failures?

Missing tool calls dominate the high-persistence BFCL condition at $\beta = 1$. We classify Qwen3-1.7B outputs by call presence and function-and-argument matches to the reference (Figure 3). Missing calls account for 89.2% of outputs at $\beta = 1$, compared with 6.9% at $\beta = 0.1$; wrong arguments account for 29.2% at $\beta = 0.5$. We analyze this pattern through the intended reference-policy constraint: penalizing departures from the reference can hinder learning necessary calls that it rarely produces.

[12] Appendix E.6 further tests whether matching responses to their original prompts detects missing tool calls.
[13] Appendix E.1 specifies the word-overlap threshold, replacement selection, and three comparisons per turn.

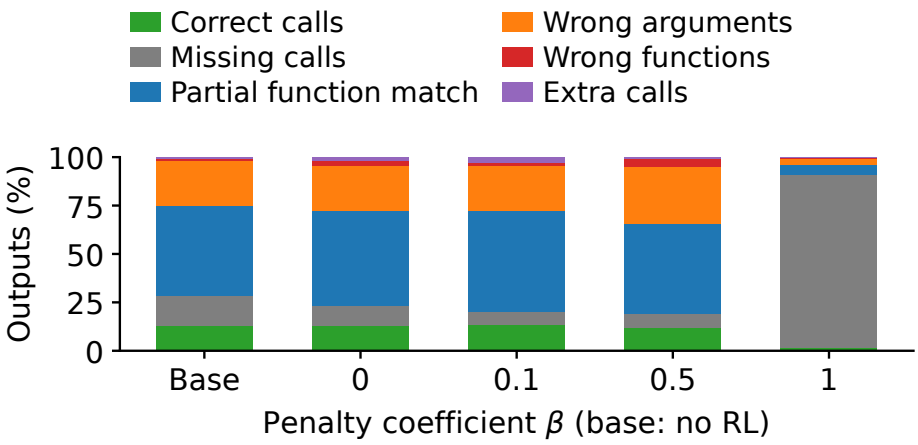

Figure 3: BFCL output error-type frequencies for Qwen3-1.7B across β.

Missing a required call also prevents the agent from obtaining its result, potentially leaving later decisions without information needed for correction and allowing failure to persist.

6.3 Does reasoning before acting improve performance?

Reasoning before acting helps both tested tasks at $\beta = 1$, but can hurt function calling at lower coefficients. We compare Qwen3-0.6B with and without pre-action reasoning text, calling the former *thinking mode*. The comparison covers the supplementary function-calling task and a single-turn GSM8K math control (Table 5).[14] At $\beta = 1$, thinking raises function-calling and math scores from 0.679 to 0.858 and from 0.383 to 0.483, respectively; at $\beta = 0$, it lowers the function-calling score from 0.819 to 0.404. Reasoning changes the context for action selection while keeping trained parameters fixed. Intermediate reasoning can clarify the required action, but can also introduce mistaken assumptions that divert an otherwise correct response.

Table 5: Task scores with standard and thinking generation for Qwen3-0.6B. FC denotes function calling; std and think denote generation without and with pre-action reasoning text.

	β=0	β=0.01	β=0.1	β=1.0
FC std	0.819	0.850	0.869	0.679
FC think	0.404	0.381	0.907	0.858
Math std	0.000	0.213	0.385	0.383
Math think	0.003	0.212	0.410	0.483

7 Conclusion

To understand how multi-turn agents propagate or recover from errors, we varied the coefficient β in training penalties motivated by KL regularization and measured the decisions immediately after failed and passed steps. We compared failure persistence and sequential consistency across five main benchmark settings, then used evaluation on the same 100 HotPotQA tasks with shuffled labels to separate failure order from overall failure frequency. Higher penalty coefficients can accompany more persistent failures, but the lowest-persistence coefficient depends on the task. Our diagnostics show that failure order can add information beyond failure frequency and that the implemented penalty can weaken reinforcement of rewarded actions. We hope this work will inform the development of reliable language-model agents that can recover from mistakes and sustain progress over extended interactions.

[14] Appendix F.2 plots the thinking-mode scores across coefficients and reports the additional entropy-bonus comparison.

AI USE STATEMENT

We used Claude and GPT to polish and revise the manuscript. These tools helped improve clarity, readability, and consistency of terminology.

REPRODUCIBILITY STATEMENT

Section 4 describes the models, benchmarks, and evaluation procedures. Appendix D.2 details gradient collection and averaging, while Appendix A.1 specifies the training settings, available software versions, and data splits. Appendix A.3 defines the pass/fail rules for each benchmark and provides labeled examples. The comparison on the same 100 HotPotQA tasks uses 600 episode records across six policies; Appendix B.2 details the transition counts, bootstrap intervals, and label-permutation tests. Appendix D.1 checks the implemented gradient; Appendices A.2, A.4, and B.3 check evaluation timing, label meaning, and metric denominators, respectively. We plan to release code, prompts, evaluation and gradient scripts, and trajectory data upon publication.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 29304–29320, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/f514cec81cb148559cf475e7426eed5e-Abstract.html.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10835–10866. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23h.html.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1856–1865. PMLR, 2018. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. Understanding the effects of RLHF on LLM generalisation and diversity. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=PXD3FAVHJT.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. *CoRR*, abs/2404.06474, 2024. doi: 10.48550/ARXIV.2404.06474. URL https://doi.org/10.48550/arXiv.2404.06474.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/e4c61f578ff07830f5c37378dd3ecb0d-Abstract-Conference.html.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. Webrl: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=oVKEAFjEqv.

Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pp. 627–635. JMLR.org, 2011. URL http://proceedings.mlr.press/v15/ross11a/ross11a.pdf.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

ByteDance Seed. UI-TARS-2 technical report: Advancing GUI agent with multi-turn reinforcement learning. *CoRR*, abs/2509.02544, 2025. doi: 10.48550/ARXIV.2509.02544. URL https://doi.org/10.48550/arXiv.2509.02544.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Llama Team. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL https://doi.org/10.48550/arXiv.2407.21783.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Zihan Wang, Chi Gui, Xing Jin, Qineng Wang, Licheng Liu, Kangrui Wang, Shiqi Chen, Linjie Li, Zhengyuan Yang, Pingyue Zhang, Yiping Lu, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN-2: reasoning collapse in agentic RL. *CoRR*, abs/2604.06268, 2026. doi: 10.48550/ARXIV.2604.06268. URL https://doi.org/10.48550/arXiv.2604.06268.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, Wei He, Yiwen Ding, Guanyu Li, Zehui Chen, Zhengyin Du, Xuesong Yao, Yufei Xu, Jiecao Chen, Tao Gui, Zuxuan Wu, Qi Zhang, Xuanjing Huang, and Yu-Gang Jiang. Agentgym-rl: Training LLM agents for long-horizon decision making through multi-turn reinforcement learning. *CoRR*, abs/2509.08755, 2025. doi: 10.48550/ARXIV.2509.08755. URL https://doi.org/10.48550/arXiv.2509.08755.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/4b175d846fb008d540d233c188379ff9-Abstract-Datasets_and_Benchmarks.html.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/82ad13ec01f9fe44c01cb91814fd7b8c-Abstract-Conference.html.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R. Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

APPENDIX OVERVIEW

The numbered appendices follow the same order as the questions in the main paper. Appendix A gives the training settings, evaluation timing, and benchmark-specific step-label rules. Appendix B defines the transition calculations and reports uncertainty, label-shuffling, and denominator checks. Appendix C reports additional model, horizon, algorithm, and training-run comparisons. Appendix D checks the implemented penalty and the gradient measurements used in the main analysis. Appendix E studies observation sensitivity, action errors, output diversity, and input dependence. Appendix F reports supervised-training, thinking-mode, and coefficient-scheduling controls. Appendix G states the evidence scope and the tests needed to examine the proposed explanation further.

A EXPERIMENTAL SETTINGS AND STEP LABELS

Appendix A.1 specifies training, software, and data settings. Appendix A.2 reports training budgets and distinguishes training-batch statistics from final-policy evaluation. Appendix A.3 defines each benchmark's step labels and gives examples. Appendix A.4 checks how those labels relate to task progress and environment feedback.

A.1 TRAINING CONFIGURATION

We specify the optimization, generation, hardware, and data settings for the initial BFCL experiments and the four-benchmark 4B experiments.

The main BFCL behavior comparison uses seed 42; the four-benchmark curves summarize 5, 2, 2, and 5 runs on HotPotQA, InterCode, ALFWorld, and WebShop, respectively. Error bars give one sample standard error, computed over defined rates; WebShop sequential consistency at $\beta = 1$ uses three defined runs.

Initial BFCL runs. The 0.6B and 1.7B runs use AdamW with learning rate 10^{-5} and zero weight decay. Each of five iterations uses 40 tasks and four rollouts per task, giving 160 rollouts. Generation uses sglang, temperature 0.7, top-p 0.95, and a 512-token limit. Training uses two NVIDIA H20 GPUs with 98 GB each. GRPO uses groups of four and computes each advantage by subtracting the group mean reward and dividing by the group reward standard deviation. It does not use a separate value model.

Four-benchmark 4B runs. The four-benchmark training implementation constructs AdamW with learning rate 10^{-5} and leaves weight decay at its default of 0.01. It creates a new optimizer at every outer iteration, resetting its state. Recorded runs use three or five iterations and 30 or 40 rollouts per iteration, as listed in Table 6. Generation uses Transformers, temperature 0.7, and a 256-token limit. Training uses one 98 GB NVIDIA H20 GPU and gradient checkpointing to reduce memory use. GRPO uses groups of two with the same within-group advantage calculation. The reference model is loaded separately to reduce peak memory use. Appendix C.1.4 lists configuration differences in the additional runs.

Software versions. The reported software environment uses Python 3.10.20, PyTorch 2.6.0+cu124, Transformers 4.51.3, NumPy 1.26.4, Tokenizers 0.21.4, and Accelerate 1.13.0. These versions are documented for the reported environment; versions for every historical run were not recorded. The exact sglang version and upstream environment revision are unavailable.

Data splits and evaluation procedure. The training/evaluation splits contain 300/100 HotPotQA, 200/100 InterCode, 150/50 ALFWorld, and 60/20 WebShop records. These are the subsets used in our experiments, rather than the full official datasets, and some runs sample only part of a subset. The additional HotPotQA checkpoint evaluation uses all 100 evaluation tasks in the same fixed order for every condition. For each episode, we parse the generated actions, apply the benchmark-specific scoring rule, and count adjacent pass/fail transitions within the episode. We aggregate these counts before computing transition rates and resample complete episodes when estimating evaluation uncertainty. We plan to release the training and evaluation code, prompts, reward and parsing logic,

Table 6: Verified training-batch sizes and outer iteration counts for the reported runs. Batch statistics are recorded before the update in each iteration.

Setting	Seeds	Rollouts/iteration	Iterations
HotPotQA	42	30	3
HotPotQA	123, 456	40	3
HotPotQA	789, 1024	40	5
InterCode, ALFWorld	42	30	3
InterCode, ALFWorld	123	40	3
WebShop	42, 123	40	3
WebShop	456, 789, 1024	40	5
PPO: HotPotQA	42	20	2
PPO: WebShop, InterCode	42	20	3
Gradient: HotPotQA	42	16	3
Llama: HotPotQA	42	30	3

trajectories, transition counts, and gradient scripts upon publication. Missing historical versions and per-seed records remain limitations.

A.2 Training Batches and Final Checkpoint Evaluation

To compare policies at a known training stage, we identify when each stored statistic was computed and which training budgets it combines. Each outer iteration first samples trajectories, then computes their rewards and transitions, and finally updates the model on those trajectories. The final training-batch statistics describe trajectories generated before the last update, whereas final-checkpoint evaluation uses the model after that update. These records therefore differ in policy parameters as well as in task sampling and decoding conditions. Table 6 reports the batch sizes and iteration counts verified from the saved records. Equal batch sizes do not by themselves establish that all historical settings were identical.

The five-seed HotPotQA and WebShop summaries mix three-iteration and five-iteration runs. They describe the available run collection, not five replications of one fixed training budget. The five seed-42 final checkpoints and the base model are evaluated on the same 100 held-out HotPotQA tasks. The HotPotQA training set contains 300 different task identifiers. This comparison fixes the task set and the seed-42 training-batch budget, but not the number of optimizer steps, since zero-advantage trajectories are skipped.

Optimizer and reference details. In the four-benchmark experiments, we create a new AdamW optimizer in each outer iteration, resetting its momentum between rollout batches. The optimizer configuration specifies the learning rate and uses the default weight decay, which is 0.01 in the verified PyTorch versions. Dependency versions are not available for every historical run. The direct-inference experiment freezes the original base reference, whereas the separate server-based trainer reloads the reference from the current model each iteration. These are distinct training procedures and should not be combined under one unqualified configuration.

Evaluation records. The HotPotQA evaluator records the task identifier, step labels, reward, step count, and whether all labels pass. Full actions, observations, final answers, stopping reasons, effective decoding settings, and exact checkpoint-version identifiers are unavailable in these records. The evaluation configuration requests temperature 0.01 and a 256-token limit, but effective generation settings and thinking-mode settings were not recorded for every run. The counterfactual records keep only the first 100 characters of each response, which is often too short to recover the tool call. These records therefore support task-matched label statistics, but not an exact replay or a complete retrospective relabeling of the generated actions.

A.3 How Each Benchmark Assigns Step Labels

A pass means that a scored step meets the benchmark-specific rule described below. It does not necessarily mean that the action is useful or that the task will succeed. We describe the implemented rules below.

BFCL. The analysis flattens all calls within a turn and compares the resulting ordered list with the reference call strings, including arguments. The lists must match exactly. Two empty lists pass; missing, extra, reordered, or mismatched calls fail. This rule differs from the official task checker, which checks the resulting state. A failed step label therefore need not imply terminal task failure.

HotPotQA. Search and Lookup return text from the supplied context paragraphs. The step passes when the observation contains at least one distinct answer token and covers at least 20% of the distinct answer tokens. Normalization lowercases the text and removes English articles and punctuation. Finish terminates the episode and passes when the answer-overlap score defined here is at least 0.5. This score is the harmonic mean of precision and recall computed from the number of distinct shared tokens, divided by the prediction and answer token-list lengths, respectively. This token-counting rule differs from official HotPotQA F1 and from exact match. Unsupported or unparseable actions fail. This answer-overlap rule may reject useful retrieval that does not contain answer tokens.

InterCode. When the task's SQLite database is available, the scorer executes the query and computes F1 over normalized result-row sets. Otherwise, it computes F1 over normalized SQL token sets. Both branches use a pass threshold of 0.5. This allows partial overlap and does not require exact SQL equivalence. Parsing and execution errors fail. For a bare `SUBMIT`, the parser returns an empty query and the scorer compares it with the reference using threshold 0.7. It therefore fails for a nonempty reference even if an earlier query was useful. The reported results retain this scoring behavior.

ALFWorld. The implementation replays stored expert observations. At step t, a predicted action passes if its lowercased, trimmed text equals or contains the expert action for that position. A step with no reference action fails. The next observation comes from the stored sequence regardless of the prediction. These results measure action matching in replay, not recovery in a live simulator. The implementation contains a goal-progress check, but this scorer does not use it.

WebShop. The adapter presents prerecorded pages. A reference match requires the same action type and either an argument substring match or sufficient shared argument words. The shared-word threshold is $\max(1, \lfloor |W_{\mathrm{ref}}|/2 \rfloor)$, where W_{ref} is the reference argument word list. If reference matching fails, the fallback accepts a valid Search or Click sharing at least one non-stopword with the goal. It also accepts Buy Now, Back to Search, and Next. If the goal is empty, any parsed Search or Click passes. Pages advance by index, and a purchase action ends the episode. These labels do not measure improvement in live product-attribute matching.

τ-bench. The extractor scores assistant tool calls and excludes ordinary assistant text and tool-result messages. A call passes when its API name and argument dictionary exactly match the next reference action. A matching name with wrong arguments fails and advances the reference pointer. For a wrong name, the extractor checks the next three reference positions for an exact match. If it finds one, skipped reference actions insert failure labels before the matched pass. Otherwise the call fails without advancing the pointer. Extra calls after the reference sequence fail. Tasks with fewer than three reference actions are excluded, and at least two scored labels are needed to contribute a transition. Because the sequence can contain inserted failures, a scored step is not always an executed call. This alignment rule can itself create repeated failures.

Examples. The following examples illustrate the rules; they are constructed examples, not sampled experimental trajectories.

- **BFCL**: an exact call match, a wrong-function call, and another exact match give (p, f, p).
- **HotPotQA**: a Search with no answer tokens, a Lookup meeting the 20% overlap threshold, and a Finish with answer-overlap score 1 give (f, p, p).

Table 7: Environment feedback and recorded task outcomes for the six evaluation settings. The feedback column describes what the agent observes; the outcome column describes what the evaluation scores.

Setting	Feedback/state transition	Recorded outcome
BFCL	Original evaluation is separate from the four-benchmark adapter.	Turn-level reference-call matching differs from the official state-based task checker.
HotPotQA	Search/Lookup selects text from the task's supplied paragraphs.	Reward is final-answer overlap, using distinct shared tokens but token-list denominators.
InterCode	SQL-token similarity replaces query execution when the task database is unavailable.	Reward is the best score reached; bare `SUBMIT` fails the step rule.
ALFWorld	Stored expert observations advance regardless of the predicted action.	Reward is 1 for all-pass episodes and half the pass fraction otherwise.
WebShop	Scripted pages advance by index until a purchase action stops the episode.	Reward is the pass fraction with a purchase bonus, not official product-match success.
τ-bench	Recorded tool calls are aligned to reference actions, with possible inserted failure labels.	Environment reward is separate from the all-pass rate of the aligned sequence.

- **InterCode**: an unparseable output, a fallback query with token-set F1 of 0.8, and a bare `SUBMIT` give (f, p, f). The last failure follows from the scorer and does not show that the preceding query was wrong.
- **ALFWorld**: matching the first expert action, issuing look instead of the second, and matching the third give (p, f, p). Stored observations still advance after look.
- **WebShop**: for a goal containing red and mug, Search with red mug, unsupported hover, and Click with Buy Now give (p, f, p).
- **τ-bench**: for distinct reference APIs A, B, C, an exact A call followed by an exact C call gives (p, f, p). The middle failure is inserted for skipped action B.

Both (p, f, p) and (f, p, f) have one pass-to-fail and one fail-to-pass transition, with no fail-to-fail transition. No transition connects different episodes.

A.4 Do the Labels Measure Task Recovery?

To test whether a return to a passing label means useful task progress, we inspect how observations change and compare the label rules with the available task evidence. Table 7 distinguishes the six settings by their actual recorded feedback. Appendix A.3 details the pass/fail rules for all six benchmarks, including parsing, matching thresholds, and labeled examples.

HotPotQA support-label check. We match the 100 evaluation tasks to the released HotPotQA validation set to obtain supporting-fact annotations. All 100 task identifiers, questions, answers, and context lists match exactly. For each annotated supporting title, we form the same title-plus-paragraph observation returned by Search and apply the existing answer-overlap rule. The rule rejects 45 of 200 annotated supporting paragraphs (22.5%) and accepts 245 of 800 context paragraphs not marked as supporting facts (30.6%). Applied to the annotated supporting sentences alone, it rejects 99 of 246 sentences (40.2%). These are disagreements with the supporting-fact annotations, not a claim that every unannotated paragraph is irrelevant. They also do not measure the label error rate on the model's actual actions, because the saved evaluation records lack those actions and observations. The check shows why an answer-overlap failure can include a useful intermediate retrieval and should not be treated as a verified task error.

Sensitivity to inserted labels in τ-bench. The available base-model subset has 29 task records, of which 24 contain at least two scored labels. Seven records contain more labels than executed tool calls, with ten inserted failure labels in total. The original calls and insertion positions are unavailable, so we cannot identify one corrected sequence. Instead, for each record we delete exactly as many failed labels as were inserted, enumerate all possible deletion positions, and pool the remaining transition

counts. This gives conservative bounds because some deletion positions may not correspond to the original alignment. Failure persistence changes from 0.840708 to a value in $[0.836538, 0.873786]$, and sequential consistency changes from 0.612245 to a value in $[0.612245, 0.652174]$. The number of adjacent transitions decreases from 162 to 152. The high failure persistence in this subset is not explained by the ten inserted labels alone, but this check does not validate the alignment as a measure of causal recovery or correct the unavailable trajectories from the comparisons across β.

Task identifiers. The InterCode task identifiers are formed from truncated question text: 200 training examples contain 161 unique identifiers, and 100 evaluation examples contain 84 unique identifiers. Seventeen identifiers occur in both splits, but no complete task record is shared when the identifier field is excluded. This is an ID-collision problem, not evidence by itself of duplicate-task leakage. Because the trainer groups rewards by task identifier, distinct questions with colliding identifiers can be normalized together if sampled in the same batch. New training runs need unique task identifiers and must retain their complete configurations and trajectories.

What remains to be validated. A direct recovery test must compare policies after a controlled error in the same task state, then check actual task progress and final success. For new HotPotQA evaluations, saving full retrieval results and final answers would allow answer-overlap labels, supporting-fact labels, and official answer scores to be compared on the same trajectory. The corpus checks and bounds above identify measurement problems; they do not replace these evaluations.

B Transition Metrics and Statistical Checks

Appendix B.1 relates step labels to all-pass probabilities and consecutive passes. Appendix B.2 reports transition counts, uncertainty intervals, label-shuffling tests, and paired comparisons from evaluation on the same 100 held-out HotPotQA tasks. Appendix B.3 checks WebShop denominators and distinguishes equal-run means from pooled rates.

B.1 How Step Labels Relate to Whole Episodes

To connect the transition identity to the supplementary tables, we first distinguish the probability that every scored step passes from the length of a run of passes. The first calculation gives the all-pass probability, and the second gives the expected length of consecutive passes under a constant continuation probability.

The probability that every step passes. Consider an episode with a fixed number T of scored steps. Define $\varepsilon_1 = P(f_1)$ and, for $t > 1$, $\varepsilon_t = P(f_t \mid p_1, \ldots, p_{t-1})$, the failure probability after all earlier steps have passed. The probability chain rule gives

$$P(p_1, \ldots, p_T) = \prod_{t=1}^{T} (1 - \varepsilon_t). \tag{4}$$

Only if these conditional failure probabilities all equal a constant ε_0 does this reduce to $(1-\varepsilon_0)^T$. This assumption is stronger than observing a similar average accuracy at each turn. Table 20 substitutes the mean per-turn error rate into this formula to illustrate its numerical scale. That substitution does not estimate the conditional probabilities in Equation 4. The formula also does not describe task success when a task can succeed after an earlier failed step.

The length of a run of passes. Let L count consecutive passing steps, starting from a pass and ending before the first failure. Suppose that after any such sequence, the next step passes with the same probability c. For an unlimited sequence and $0 \leq c < 1$, this assumption gives

$$\mathbb{E}[L \mid p_1] = \sum_{k=1}^{\infty} P(L \geq k \mid p_1) = \sum_{k=1}^{\infty} c^{k-1} = \frac{1}{1-c}. \tag{5}$$

For illustration, inserting the BFCL consistency values 0.042, 0.130, and 0.143 gives expected lengths of 1.04, 1.15, and 1.17 consecutive passes, respectively. These are model-based calculations, not measured episode lengths or task horizons. Finite episodes and changes in step difficulty can break the assumptions behind this summary.

B.2 Episode Bootstrap and Shuffled Labels

Let N_{ab} count adjacent scored labels from a to b within episodes, pooled over tasks, for $a, b \in \{f, p\}$. The estimates used throughout the paper are

$$\widehat{P}(f|f) = \frac{N_{ff}}{N_{ff} + N_{fp}}, \qquad \widehat{P}(p|p) = \frac{N_{pp}}{N_{pf} + N_{pp}}.$$

Observed recovery is $N_{fp}/(N_{ff} + N_{fp})$; empty conditioning sets give undefined rates, and no pair joins different episodes or extends beyond termination.

To check how much the transition estimates depend on the sampled tasks and label order, we reanalyze all six HotPotQA conditions. The base model and five seed-42 Qwen3-4B checkpoints use the same 100 task identifiers in the same order. We verified no task-identifier or normalized-question overlap with the 300 training tasks. The evaluator requests temperature 0.01, at most 256 generated tokens per step, and at most eight steps. The client samples at positive temperature, so this is not strictly greedy decoding. The evaluation records omit the effective decoding configuration, random-number state, and exact checkpoint-version identifiers. Thus, task identity is verified, while the full historical generation protocol is only partly recoverable.

The training code saves checkpoints after the last update, whereas the rates from the final batch of training trajectories in Table 1 are measured before that update. Different model states, tasks, and decoding settings prevent merging these two sources into one experiment. For each held-out evaluation, we pool within-episode transition counts and resample 100 whole episodes with replacement 5,000 times using a fixed analysis seed. The 2.5th and 97.5th percentiles of defined replicates give the 95% confidence intervals (CIs). A replicate with an empty conditioning set is undefined and excluded from that metric's interval, with the excluded count reported below. This replaces the earlier analysis convention of assigning zero to an empty denominator. Resampling whole episodes preserves their internal order but does not measure variation across training seeds.

Table 8: Failure persistence on 100 held-out HotPotQA episodes per condition. Brackets give 95% episode-bootstrap confidence intervals.

Condition	P(f|f)	95% CI	N_{ff}	$N_{ff} + N_{fp}$
Base	.869	[.827, .904]	372	428
$\beta = 0.0$	.986	[.977, .994]	648	657
$\beta = 0.01$	.988	[.979, .995]	640	648
$\beta = 0.1$	.956	[.935, .973]	584	611
$\beta = 0.5$	.988	[.976, .997]	669	677
$\beta = 1.0$	.962	[.943, .979]	560	582

The paired comparisons below do not establish equivalent performance across the tested coefficients. Overlapping individual intervals would not by themselves establish equivalence either.

Sequential consistency and all-pass rate. We also recompute sequential consistency and the fraction of episodes in which every scored step passes. The latter is the all-pass rate, not task success. We use 95% Wilson binomial confidence intervals for this binary episode outcome, including conditions with zero observed all-pass episodes.

Rare pass transitions make consistency estimates uncertain. For P(p|p), one replicate out of 5,000 is undefined at each of $\beta = 0.01$ and $\beta = 0.5$; all other individual transition-rate replicates are defined. For zero all-pass episodes out of 100, the Wilson interval is $[0, .037]$ rather than the degenerate empirical-bootstrap interval $[0, 0]$. We also report the answer-overlap threshold outcome with Wilson binomial intervals, separately from all-pass rate. For example, the base model reaches the answer threshold on 28 episodes but passes every step on only six. These intervals cover this checkpoint evaluation, not every benchmark curve or training seed. We do not infer missing intervals from rounded means.

Shuffling within each episode. The control uniformly reorders pass/fail labels within each episode 5,000 times, preserving its length, pass count, and all-pass outcome. The stored answer reward is

Table 9: Sequential consistency and all-pass rate on the same HotPotQA episodes. Intervals use episode bootstrap for consistency and Wilson scores for all-pass rate.

Condition	P(p|p)	95% CI	N_{pp}	N_{pf}	All-pass	95% CI
Base	.446	[.317, .562]	33	41	.060	[.028, .125]
$\beta = 0.0$	.188	[.000, .429]	3	13	.000	[.000, .037]
$\beta = 0.01$	.500	[.083, .771]	10	10	.000	[.000, .037]
$\beta = 0.1$	.342	[.212, .452]	13	25	.000	[.000, .037]
$\beta = 0.5$	.143	[.000, .278]	2	12	.000	[.000, .037]
$\beta = 1.0$	.468	[.200, .646]	29	33	.020	[.006, .070]

also unchanged. Table 10 reports the shuffled mean and the difference from the shuffled baseline with episode-bootstrap uncertainty. To compute the excess efficiently in every bootstrap replicate, we subtract the ratio of expected shuffled transition counts from the observed conditional rate. For an episode of length T with n_f failures and n_p passes, these expectations are $\mathbb{E}[N_{ff}] = n_f(n_f - 1)/T$ and $\mathbb{E}[N_{fp}] = n_f n_p/T$. We sum the expected counts across episodes before taking their ratio. This ratio differs slightly from averaging the conditional rate over sampled permutations, so the table distinguishes the two quantities.

Table 10: Within-episode shuffle control for HotPotQA failure persistence. Shuffled mean averages sampled permutations; Excess subtracts the ratio of expected shuffled counts from the observed rate. Brackets give 95% episode-bootstrap intervals for Excess.

Condition	Observed	Shuffled mean	Excess	95% CI for excess
Base	.869	.879	$-.0095$	$[-.0270, +.0080]$
$\beta = 0.0$	.986	.981	$+.0058$	$[+.0008, +.0119]$
$\beta = 0.01$	.988	.981	$+.0068$	$[+.0005, +.0149]$
$\beta = 0.1$	.956	.950	$+.0053$	$[-.0034, +.0151]$
$\beta = 0.5$	.988	.983	$+.0054$	$[+.0005, +.0114]$
$\beta = 1.0$	.962	.949	$+.0135$	$[+.0015, +.0260]$

Most of the high absolute failure persistence remains after shuffling. The control tests whether label order matters beyond the failure count, but cannot attribute the remaining dependence solely to earlier model errors. Task stage, observation construction, and scoring rules can also affect label order. For each condition and each transition metric, the one-sided permutation test asks whether the observed rate exceeds its shuffled distribution. We use $p = (1 + B_{\geq})/(5001)$, where $B_{\geq}$ counts permutations at least as large as the observed rate, and apply the Holm multiple-testing procedure across all 12 tests. All permutation replicates have defined conditioning sets. Table 11 includes both metrics and every condition, including the base model and $\beta = 0.01$. Only $\beta = 1.0$ has excess failure persistence at adjusted $p < 0.05$; the base model and $\beta = 0.1$ have excess sequential consistency under the same correction. The unadjusted excess CIs and the corrected permutation tests answer different questions and should not be read as interchangeable significance statements.

Table 11: One-sided permutation tests for both transition rates across the six HotPotQA conditions. Raw and Holm-adjusted p-values are reported.

	P(f|f)		P(p|p)	
Condition	Raw p	Holm p	Raw p	Holm p
Base	.8862	1.0000	.0040	.0400
$\beta = 0.0$	.0180	.1260	.3703	1.0000
$\beta = 0.01$	.0062	.0558	.0264	.1584
$\beta = 0.1$	.1158	.5789	.0010	.0110
$\beta = 0.5$	.0140	.1120	.6443	1.0000
$\beta = 1.0$	.0004	.0048	.7315	1.0000

All paired comparisons. For every pair of conditions, we resample the same task identifiers together and recompute the difference in pooled rates. Table 12 reports all 15 pairs rather than selecting pairs by a post-hoc accuracy threshold. These intervals are descriptive and are not adjusted for comparisons across pairs. For consistency differences, one replicate is undefined for pairs containing only one of $\beta = 0.01$ and $\beta = 0.5$, and two are undefined for their joint comparison. The two coefficients $\beta = 0.1$ and $\beta = 1.0$ both reach the answer-overlap threshold on five of 100 tasks, but neither transition-rate difference has an interval excluding zero. This comparison does not establish an additional recovery benefit at matched answer performance.

Table 12: Paired differences between all conditions in the reported HotPotQA evaluation. Each difference is condition B minus condition A with a 95% paired episode-bootstrap interval.

A	B	ΔP(f|f)	95% CI	ΔP(p|p)	95% CI
Base	0	$+.117$	$[+.082, +.160]$	$-.258$	$[-.507, +.020]$
Base	.01	$+.118$	$[+.083, +.161]$	$+.054$	$[-.378, +.362]$
Base	.1	$+.087$	$[+.048, +.130]$	$-.104$	$[-.263, +.050]$
Base	.5	$+.119$	$[+.083, +.162]$	$-.303$	$[-.500, -.110]$
Base	1	$+.093$	$[+.055, +.138]$	$+.022$	$[-.269, +.248]$
0	.01	$+.001$	$[-.010, +.012]$	$+.312$	$[-.154, +.650]$
0	.1	$-.030$	$[-.051, -.012]$	$+.155$	$[-.108, +.388]$
0	.5	$+.002$	$[-.010, +.014]$	$-.045$	$[-.320, +.200]$
0	1	$-.024$	$[-.045, -.005]$	$+.280$	$[-.075, +.552]$
.01	.1	$-.032$	$[-.054, -.012]$	$-.158$	$[-.436, +.259]$
.01	.5	$+.001$	$[-.011, +.011]$	$-.357$	$[-.688, +.065]$
.01	1	$-.025$	$[-.047, -.007]$	$-.032$	$[-.437, +.429]$
.1	.5	$+.032$	$[+.011, +.055]$	$-.199$	$[-.394, -.015]$
.1	1	$+.006$	$[-.016, +.030]$	$+.126$	$[-.177, +.362]$
.5	1	$-.026$	$[-.047, -.006]$	$+.325$	$[+.034, +.561]$

Matching a task across two conditions by its realized episode length and pass count leaves only zero to five mixed-label pairs per comparison. Most matched episodes contain only failures, and selecting by these policy-dependent outcomes is not a controlled recovery test. The shuffle analysis establishes sensitivity to label order beyond marginal pass counts, but does not show improved prediction of task success or gains from selecting β with transition metrics.

Analysis inputs and outputs. The analysis uses 600 episode records: the same 100 tasks evaluated under six conditions, each with a task identifier, ordered step labels, and a reward. We align conditions by task identifier, count transitions within each episode, and apply the resampling and permutation procedures above. We report step-pass rates, answer-overlap outcomes, all pairwise comparisons, and the number of resamples with undefined transition rates. The records contain evaluation outcomes rather than complete actions, observations, or answer strings, so this analysis operates on the measured labels and rewards.

B.3 WebShop Conditional-Denominator Audit

To distinguish a measured zero rate from an empty conditioning set, we reconstruct the four transition counts for each of the 25 reported WebShop training batches. The available summaries store conditional rates, total transitions, episode counts, accuracy by step position, and the all-pass rate, but not the individual trajectories. The counts below are therefore reconstructed from aggregates rather than read from raw episodes.

Reconstruction. We enumerate nonnegative integer transition counts that sum to the stored transition total and reproduce all four conditional rates under the evaluation rule's zero-denominator convention. We also enumerate nonincreasing episode counts by step position that reproduce the stored position-wise accuracies and total number of labels. The number of passing labels after the first position must equal $N_{fp} + N_{pp}$. The stored all-pass rate constrains the number of episodes containing any failed label, which must lie between the largest position-specific failure count and the

Table 13: Uniquely reconstructed aggregate WebShop transition counts from the final training batch of each run. $N_f = N_{ff} + N_{fp}$ and $N_p = N_{pf} + N_{pp}$ are the conditioning counts for the two rates.

β	Seed	N_{ff}	N_{fp}	N_{pf}	N_{pp}	N_f	N_p
0	42	0	2	4	114	2	118
0	123	0	7	14	91	7	105
0	456	0	9	1	110	9	111
0	789	2	3	4	75	5	79
0	1024	2	17	19	82	19	101
0.01	42	1	3	24	92	4	116
0.01	123	4	1	5	87	5	92
0.01	456	14	28	17	47	42	64
0.01	789	0	4	4	109	4	113
0.01	1024	0	3	5	104	3	109
0.1	42	0	4	2	101	4	103
0.1	123	0	3	13	98	3	111
0.1	456	0	1	1	78	1	79
0.1	789	63	24	28	5	87	33
0.1	1024	5	9	8	96	14	104
0.5	42	2	10	3	86	12	89
0.5	123	0	28	24	20	28	44
0.5	456	2	4	5	72	6	77
0.5	789	85	17	11	4	102	15
0.5	1024	0	7	10	82	7	92
1.0	42	75	22	22	1	97	23
1.0	123	15	30	37	38	45	75
1.0	456	88	10	20	2	98	22
1.0	789	120	0	0	0	120	0
1.0	1024	120	0	0	0	120	0

total number of failed labels. Exactly one four-cell transition table satisfies these constraints for each of the 25 records. This uniqueness is conditional on the stored aggregates and tracker convention being correct; it does not recover episode identities or validate the step labels.

All 25 failure-persistence denominators are positive. The three zero values at $\beta = 0$ use 2, 7, and 9 failure-conditioned transitions, respectively, so they are measured zeros rather than undefined rates. In contrast, $\beta = 1$ with seeds 789 and 1024 has no pass-conditioned transitions, making sequential consistency undefined for those two runs. Their recorded consistency zeros are replaced by NA, while their failure-persistence values remain included.

Averaging and uncertainty. Table 14 averages only defined rates, with each run receiving equal weight. SE is the sample standard deviation divided by the square root of the number of defined runs. Failure-persistence means remain 0.101053 at $\beta = 0$ and 0.800898 at $\beta = 1$, a difference of 0.699845. Excluding the two undefined consistency values changes the $\beta = 1$ consistency mean from 0.128211 to 0.213685, based on three defined runs.

The pooled failure-persistence fractions are $4/42$, $19/58$, $68/109$, $89/155$, and $418/480$ in increasing coefficient order. These pooled rates weight runs by their conditional support, so they answer a different question from the equal-run mean. The difference is large at $\beta = 0.1$ and 0.5, where failure-conditioned counts vary sharply across runs. A valid zero based on one or two transitions remains a very imprecise estimate. We do not treat the pooled transitions as independent trials or infer an episode-bootstrap interval from aggregates without episode records. The audited increase is a descriptive result for scripted-page training batches with mixed budgets, not a verified effect on final-policy recovery in a live WebShop environment.

Table 14: WebShop run means after checking conditional denominators. Run counts give the numbers of defined rates; pooled failure persistence uses summed transition counts.

β	$P(f \mid f) \pm$ SE	$P(p \mid p) \pm$ SE	Defined runs (f/p)	Pooled $P(f \mid f)$
0	0.101 ± 0.077	0.917 ± 0.034	5/5	0.095
0.01	0.277 ± 0.147	0.878 ± 0.048	5/5	0.328
0.1	0.216 ± 0.145	0.785 ± 0.160	5/5	0.624
0.5	0.267 ± 0.155	0.703 ± 0.143	5/5	0.574
1.0	0.801 ± 0.124	0.214 ± 0.147	5/3	0.871

C Additional Behavioral Results

Appendix C.1 collects results across training runs. Appendix C.2 reports BFCL transition rates, per-turn accuracy, and episode summaries. Appendix C.3 extends the comparisons to model size, turn budget, PPO, and Llama. Appendix C.4 compares the best tested task scores in the supplementary 0.6B experiments.

C.1 Results Across Training Runs

To check how much the reported ordering changes across runs, we compare base and trained models and list the available seed summaries. Appendix C.1.1 compares failure persistence before and after RL training. Appendix C.1.2 reports the HotPotQA results for five training seeds. Appendix C.1.3 reports the corresponding WebShop results and explains how undefined rates are handled. Appendix C.1.4 adds the reported five-run summaries for InterCode and ALFWorld.

C.1.1 Before and After RL Training

Table 15 compares the base model with seed-42 runs; its HotPotQA values are individual-run training-batch rates, not the five-run means or the held-out results.

Table 15: Failure persistence before and after training on BFCL (1.7B) and HotPotQA (4B). Δ is the trained rate minus the base-model rate.

Condition	Base P(f|f)	Trained P(f|f)	Δ
BFCL (Qwen3-1.7B)			
β=0	0.859	0.833	−0.026
β=0.1	0.859	0.841	−0.018
β=0.5	0.859	0.868	+0.009
β=1.0	0.859	0.994	+0.135
HotPotQA (Qwen3-4B)			
β=0	0.886	0.871	−0.015
β=0.1	0.886	0.898	+0.013
β=0.5	0.886	0.785	−0.101
β=1.0	0.886	0.884	−0.002

Training can increase or decrease failure persistence under the same environment implementation. The increase is 0.135 for BFCL at $\beta = 1.0$, whereas HotPotQA at $\beta = 0.5$ decreases by 0.101. The base-model rate reflects the model's own errors as well as task and environment effects. These comparisons also do not hold the visited states fixed or identify template collapse as the cause of a change.

C.1.2 HotPotQA

Table 16 gives the available Qwen3-4B results for seeds 42, 123, 456, 789, and 1024.

Table 16: HotPotQA failure persistence by training seed. The final column gives the mean and sample standard error across the five seeds; bold marks the lowest mean.

β	s42	s123	s456	s789	s1024	Mean±SE
0.0	.871	.865	.766	.845	.850	.839±.019
0.01	.939	.834	.920	.890	.716	.860±.040
0.1	.898	.880	.757	.615	.669	**.764±.056**
0.5	.785	.891	.774	.667	.769	.777±.036
1.0	.884	.762	.833	.688	.913	.816±.041

The lowest mean is 0.764 at $\beta = 0.1$, followed by 0.777 at $\beta = 0.5$. The mean of 0.816 at $\beta = 1.0$ is below the means at $\beta = 0$ and $\beta = 0.01$. Thus, the high-β endpoint is not the worst condition in this table.

All reported means and SEs use five runs per condition. The SEs summarize runs with different training budgets and do not establish either a unique optimum or equivalent performance across nearby coefficients. Episode bootstrapping instead measures variation across evaluation tasks for fixed checkpoints; Appendix B.2 details this procedure on the same 100 HotPotQA tasks for all six policies. We do not combine benchmark ranks into a significance test because the benchmarks share models and code and use different labels and sample sizes.

C.1.3 WebShop

Table 17 reports the five-seed WebShop results.

Table 17: WebShop failure persistence by training seed. The final column gives the mean and sample standard error over defined rates; bold marks the highest mean.

β	s42	s123	s456	s789	s1024	Mean±SE
0.0	.000	.000	.000	.400	.105	.101±.077
0.01	.250	.800	.333	.000	.000	.277±.147
0.1	.000	.000	.000	.724	.357	.216±.145
0.5	.167	.000	.333	.833	.000	.267±.155
1.0	.773	.333	.898	1.00	1.00	**.801±.124**

The mean increases from 0.101 at $\beta = 0$ to 0.801 at $\beta = 1.0$. Individual seeds still vary, so the mean ordering is not an ordering that holds in every run. Every failure-persistence denominator is positive, including all entries shown as zero. Thus, the zero values are valid observations under the replay scorer, although some are based on very few fail-conditioned transitions. Two sequential-consistency denominators at $\beta = 1$ are empty; excluding them changes its mean from the zero-filled value 0.128 to 0.214 over three defined runs. Appendix B.3 gives the uniquely reconstructed aggregate counts conditional on the reported summaries, not recovered episode trajectories.

C.1.4 Additional InterCode and ALFWorld Runs

Table 18 reproduces the reported five-run means and SEs for seeds 42, 123, 456, 789, and 1024.

The specified configuration for the three added seeds uses three iterations, ten tasks per iteration, and two rollouts per task. The original seed-42 and seed-123 summaries have final batches of 30 and 40 episodes, respectively. The SE therefore includes configuration differences as well as variation across seeds. The added per-seed records are unavailable, so we do not derive paired tests or further intervals from these rounded summaries.

Table 18: Expanded InterCode and ALFWorld failure-persistence results across five seeds. Entries are mean ± standard error; bold marks the highest mean.

Benchmark	β	Mean ± SE
InterCode	0.0	.894 ± .035
	0.1	.843 ± .028
	0.5	.820 ± .063
	1.0	**.917** ± .031
ALFWorld	0.01	.732 ± .012
	0.1	.730 ± .034
	0.5	.763 ± .037
	1.0	**.856** ± .030

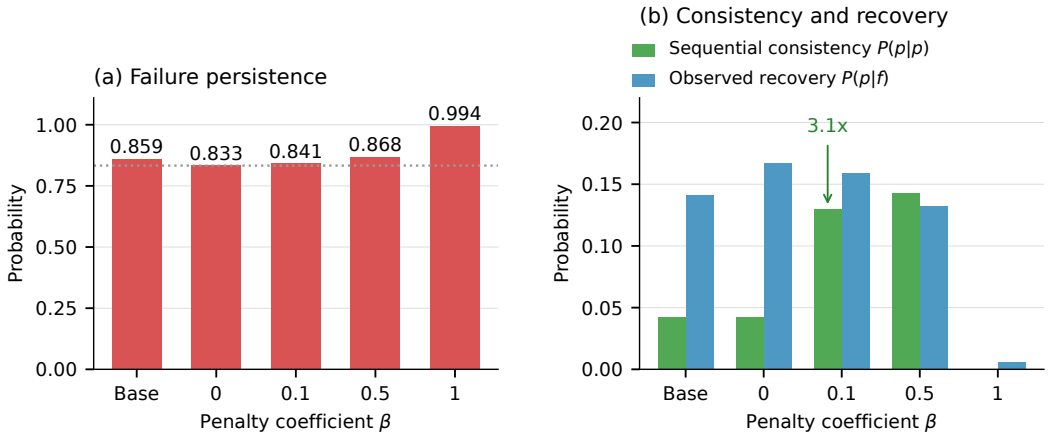

Figure 4: BFCL transition rates for Qwen3-1.7B: failure persistence (left), and sequential consistency and observed recovery (right). The dotted line marks failure persistence at $\beta = 0$; the arrow marks the 3.1-fold increase in consistency from $\beta = 0$ to 0.1.

C.2 BFCL Transition Results

Appendix C.2.1 reports BFCL failure persistence, sequential consistency, and observed recovery across β. Appendix C.2.2 breaks accuracy down by turn to locate where performance changes. Appendix C.2.3 illustrates the all-pass calculation from Appendix B.1.

C.2.1 Failure Persistence and Sequential Consistency

Figure 4 and Table 19 report Qwen3-1.7B BFCL results for seed 42.

Table 19: BFCL transition summaries (Qwen3-1.7B, seed 42). Observed recovery is $1 - P(\text{f}|\text{f})$; bold marks the highest value in each of the first two rate columns.

β	$P(\text{f}\|\text{f})$	$P(\text{p}\|\text{p})$	Observed recovery
Base	.859	.042	.141
0.0	.833	.042	.167
0.1	.841	.130	.159
0.5	.868	**.143**	.132
1.0	**.994**	.000	.006

The largest recorded recovery rate is 0.167 at $\beta = 0$. The largest sequential consistency is 0.143 at $\beta = 0.5$. These two summaries need not prefer the same coefficient. Their denominators also differ, which matters when passes are rare.

C.2.2 Per-Turn Accuracy

The BFCL turn decomposition locates where performance differs across coefficients (Figure 5). For Qwen3-1.7B, accuracy at $\beta = 1$ reaches zero by turn 2, while several other settings improve at later turns. These turn averages locate failures but do not show that earlier errors caused them.

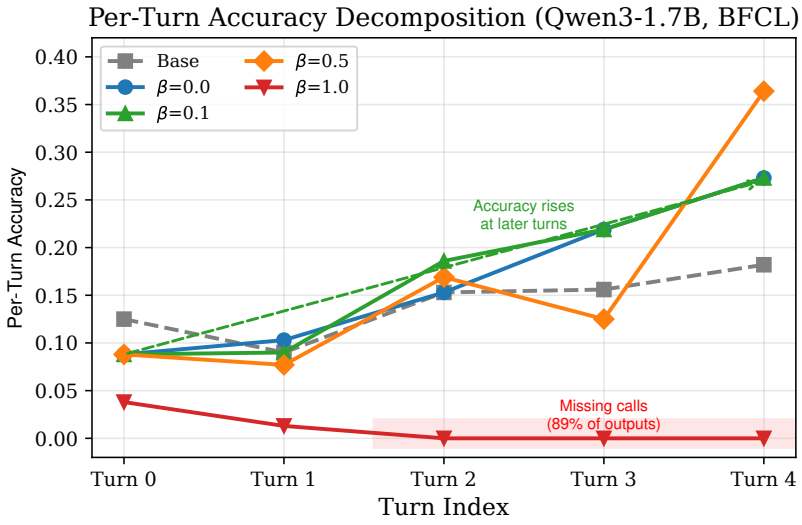

Figure 5: BFCL accuracy at each zero-indexed turn for Qwen3-1.7B. The missing-call annotation summarizes all turns at $\beta = 1$, rather than a single turn.

C.2.3 An All-Pass Calculation

Table 20 inserts $\varepsilon_0 = 1 -$ mean per-turn accuracy and $T = 5$ into $(1 - \varepsilon_0)^T$. As explained in Appendix B.1, the measured mean is not the conditional error probability required by that formula.

Table 20: Illustrative all-pass probabilities for BFCL. Formula estimates all-pass probability using $(1 - \varepsilon_0)^5$; the recorded task pass rate is shown separately.

Condition	ε_0	Formula	Recorded task pass
Base	0.859	0.0001	0.000
β=0	0.833	0.0001	0.000
β=0.1	0.829	0.0001	0.000
β=0.5	0.835	0.0001	0.000
β=1.0	0.990	0.0000	0.000

An error rate near 0.83 corresponds to about 17% per-turn accuracy. With such low accuracy, the formula is near zero for all displayed conditions and does little to distinguish them.

C.3 Model Size, Turn Budget, and Training Algorithm

To test the limits of the main comparison, we add results at a larger model size, with a larger turn budget, and under PPO. Appendix C.3.1 reports the Qwen3-8B HotPotQA comparison. Appendix C.3.2 examines the effect of increasing the maximum turn budget. Appendix C.3.3 describes the retail tasks and reports transition counts for shorter and longer scored sequences. Appendix C.3.4 compares

PPO results on HotPotQA, WebShop, and InterCode. Appendix C.3.5 reports the Llama-3.1-8B comparison.

C.3.1 THE 8B EXPERIMENT

The Qwen3-8B HotPotQA experiment uses stochastic gradient descent (SGD) so that the policy and reference model fit on one GPU. Each run has three iterations, with 15 tasks and two rollouts per task. Failure persistence ranges from 0.848 at $\beta = 1.0$ to 0.882 at $\beta = 0.1$. The range across the tested coefficients is 0.034. Here $\beta = 1.0$ has the lowest rate, so this experiment does not support a common ranking across model sizes. Model size and optimizer both change, which prevents attributing the difference to size alone.

C.3.2 A LARGER TURN BUDGET

We increase the maximum HotPotQA turn budget from 8 to 20. Table 21 reports the mean realized length (Turns/ep), which remains between 5.1 and 6.9 turns, together with the mean answer-overlap reward.

Table 21: HotPotQA results with a maximum of 20 turns (Qwen3-4B). Δ is 20-turn-budget failure persistence minus the corresponding standard-budget value.

β	$P(\text{f}\mid\text{f})$	Turns/ep	Reward	Δ vs. std
0.0	0.859	6.9	0.625	-0.085
0.1	0.708	5.1	0.771	-0.109
0.5	0.706	5.6	0.542	-0.140
1.0	0.857	6.1	0.463	-0.041

Failure persistence decreases by 0.109 and 0.140 at $\beta = 0.1$ and $\beta = 0.5$, compared with 0.041 at $\beta = 1.0$. The nonzero observed recovery at $\beta = 1.0$ shows that failed steps can still be followed by passing steps. These summaries describe behavior with a larger budget, but the available records do not establish a comparison with all other training conditions held fixed. It also does not show which coefficient would work best for tasks lasting hundreds of turns.

C.3.3 τ-BENCH RETAIL

The retail tasks require an agent to check user identity, retrieve order or product information, and change orders through API calls (Yao et al., 2025). The scored sequences contain 2–18 steps, with a median of 8, while the full conversations contain 10–62 messages. Scored steps can include failures inserted when executed calls skip reference actions; Appendix A.3 details this call-alignment rule and gives a labeled example. We serve Qwen3-8B through vLLM with the Hermes tool-call parser, thinking disabled, and temperature 0. The base model is evaluated on 115 tasks, and the GRPO-SGD conditions at $\beta = 0$ and $\beta = 1.0$ each use 30 tasks. Table 22 gives the base-model transition counts after filtering.

Table 22: Base-model τ-bench transition counts from the 115-task evaluation. Columns report the four pass/fail transition counts; T is the scored sequence length.

Horizon	ff	fp	pf	pp
Short ($T \leq 8$)	88	23	28	41
Long ($T > 8$)	230	39	43	41
Overall	318	62	71	82

C.3.4 PPO ACROSS THREE BENCHMARKS

We compare failure persistence across β under PPO, starting with HotPotQA and then extending to WebShop and InterCode.

The HotPotQA PPO experiment uses Qwen3-4B, two iterations, ten tasks per iteration, and two rollouts per task. Each batch is used for three PPO epochs with clipping parameter $\epsilon = 0.2$. Clipping limits changes in the ratio between new and rollout-policy action probabilities. The clip fraction in Table 23 is the fraction of ratios affected by this limit.

Table 23: HotPotQA failure persistence under PPO and GRPO with the same task count and iteration count. Reward is the mean final-answer overlap score; clip fraction summarizes PPO probability-ratio clipping.

β	$P(\text{f}\mid\text{f})$	Reward	Clip frac.	Algorithm
0.0	0.890	0.252	—	PPO
0.1	0.913	0.088	0.17	PPO
0.5	0.852	0.108	0.17	PPO
1.0	0.841	0.213	0.17	PPO
GRPO comparison (2 iterations, 10 tasks per iteration)				
0.5	0.817	—	—	GRPO
1.0	0.898	—	—	GRPO

All four PPO conditions have failure persistence above 0.84, but $\beta = 1.0$ has the lowest rate. The added WebShop and InterCode PPO experiments have different orderings (Table 24). Clipping may contribute to these differences, but the experiments do not isolate its effect. The results show repeated failures under PPO, not a coefficient ranking that transfers from GRPO.

Table 24: PPO failure persistence on the additional WebShop and InterCode experiments with Qwen3-4B and seed 42. Rates come from the final batch of training trajectories sampled for the update.

Benchmark	$\beta = 0.0$	$\beta = 0.1$	$\beta = 0.5$	$\beta = 1.0$
WebShop	.629	.898	1.000	1.000
InterCode	.990	.980	.936	1.000

Additional PPO configurations. The WebShop and InterCode extensions use Qwen3-4B, seed 42, three iterations, ten tasks per iteration, and two rollouts per task. They use three PPO epochs, clipping parameter 0.2, AdamW learning rate 10^{-5}, gradient clipping 1.0, temperature 0.7, and a 160-token limit. The final batches contain 20 training rollouts per condition, with 60 WebShop transitions or 100 InterCode transitions. These summarize training trajectories, not held-out evaluations after the final update. The HotPotQA PPO comparison above uses two iterations. The benchmark rows therefore do not form a fully matched comparison.

C.3.5 LLAMA-3.1-8B

The Llama-3.1-8B-Instruct HotPotQA experiment uses seed 42 and three GRPO-SGD iterations, with 15 tasks and two rollouts per task. The learning rate is 5×10^{-6}, gradient clipping is 1.0, temperature is 0.7, and the generation limit is 256 tokens. Final batches contain 30 episodes and 155, 148, 145, and 145 transitions for $\beta = 0, 0.1, 0.5, 1.0$, respectively. Table 25 reports both behavioral rates for this non-Qwen comparison. Failure persistence is lowest at $\beta = 1$, whereas sequential consistency is highest at $\beta = 0.5$, again separating observed recovery from maintaining a pass.

C.4 WHICH TESTED COEFFICIENT PERFORMS BEST?

To avoid treating a best tested coefficient as a universal optimum, we list the task-success scores and compare their task settings. Table 26 reports the best tested task-success scores in the initial Qwen3-0.6B experiments. Here β^* denotes the best tested coefficient, rather than an optimum over all possible values. The Code and Math controls use MBPP, a short Python-programming benchmark (Austin et al., 2021), and GSM8K, a grade-school math benchmark (Cobbe et al., 2021), respectively.

Table 25: Llama-3.1-8B-Instruct results on HotPotQA with GRPO-SGD. Rates use the final training batch; bold marks the lowest failure persistence and highest sequential consistency.

β	$P(f\|f)$	$P(p\|p)$
0.0	.846	.627
0.1	.851	.617
0.5	.828	**.673**
1.0	**.742**	.536

Table 26: Initial Qwen3-0.6B task-success comparison. FC denotes BFCL, Code denotes MBPP, Web denotes the WebShop-style control, and Math denotes GSM8K. Δ is the displayed best score minus the base score, and T is the reported task length.

Domain	β^*	Best	Base	Δ	Reported T
FC	0.05	0.893	0.590	+0.303	4–5
Code	0.05	0.416	0.113	+0.303	$\sim$1
Web	0.01	0.668	0.015	+0.653	1–3
Math	0.5	0.415	0.365	+0.050	$\sim$1

Task differences. The coefficient that minimizes failure persistence need not maximize task success. BFCL has its lowest failure persistence at $\beta = 0$ in Table 19, whereas HotPotQA has its lowest mean at $\beta = 0.1$ in Table 16. These tasks differ in difficulty, action format, labels, and typical length. Their best coefficients therefore cannot establish a relationship between task length and the best β. The larger-budget HotPotQA results in Appendix C.3.2 describe episode lengths and failure persistence with a 20-turn limit, but do not isolate turn budget from other training differences. Neither comparison proves that performance must peak at an interior coefficient.

D Training Loss and Gradient Measurements

Appendix D.1 derives the sampled-penalty gradient and checks it numerically. Appendix D.2 details gradient sampling, parameter selection, and averaging. Appendix D.3 reports parameter distance from initialization.

D.1 What Gradient Does the Implemented Penalty Supply?

To determine whether the coefficient experiments measure the intended KL objective, we differentiate the loss used by the four-benchmark training implementation. The trainer takes a fixed trajectory advantage A, averages log probabilities over sampled assistant tokens, and adds $\beta(\bar{\ell}_\theta - \bar{\ell}_{\text{ref}})$ to $-A\bar{\ell}_\theta$. Both terms use the same assistant-token mask and token-count denominator. The reference calculation has no gradient, and the sampled tokens are treated as fixed during the update. Equation 2 therefore follows directly from this implementation. Changing the reference probabilities changes the logged scalar penalty but not its gradient on those fixed tokens. This derivative does not implement the gradient of the intended KL objective.

Numerical check. We check the fixed-token calculation with policy scores $(0.7, -0.2, 0.1)$ before probability normalization, the first token as the observed target, $A = 1$, and $\beta = 0.5$. Replacing reference scores $(0.2, 0.3, -0.4)$ with $(-0.8, 0.9, 0.5)$ changes the loss from 0.823323 to 1.493697. Central finite differences with step size 10^{-6} give a maximum gradient difference of 1.11×10^{-10} between the two references. The maximum difference from the analytic gradient is 4.68×10^{-11}. This small calculation checks the loss derivative, not model behavior after training.

Consequences for the gradient measurements. We compute the gradient measurements using the same task and sampled-penalty losses as in training. For trajectory j, let $g_{\text{pg},j}$ and $g_{\text{KL},j}$ be their gradients before multiplying the penalty by β, and let A_j be its advantage. The implementation gives

$$g_{\text{pg},j} = -A_j g_{\text{KL},j}. \tag{6}$$

Thus, the two gradients for one trajectory share the same direction up to the scalar $-A_j$. Their means across trajectories can have different directions because the advantages weight trajectories differently. The reported penalty-gradient ratio and cosine describe these averages, but they are not independent evidence that a reference-dependent KL gradient opposes the task gradient. With two unequal rewards in a group, normalized advantages are approximately $+1$ and -1; at $\beta = 1$, the sampled loss nearly cancels the positive-advantage trajectory's gradient and increases the negative-advantage trajectory's gradient magnitude. This statement concerns the two loss terms before clipping and optimization, and does not include AdamW weight decay.

Scope. The four-benchmark GRPO experiments, the SGD variants, the PPO experiments, and the gradient measurements use the same sampled log-ratio penalty. PPO uses a clipped policy-ratio term for its task loss, but its sampled penalty still has the reference-independent derivative described above. The original BFCL experiments used a separate training implementation whose loss is not established by this derivation. Testing the intended KL effect requires controlled training with a verified KL gradient.

D.2 Gradient Measurements

We measure gradients on Qwen3-4B during HotPotQA GRPO training at $\beta \in \{0, 0.1, 0.5, 1.0\}$. Each run has three iterations, with eight tasks and two rollouts per task in each iteration. Before each update, we use automatic differentiation to compute the task-loss and sampled-penalty gradients separately for each trajectory. We measure the final decoder block, output head, and layer norm. This keeps the computation manageable and covers parameters close to the output, but does not establish how earlier layers behave.

Only tokenizable trajectories with nonzero advantage enter the gradient calculation; an iteration without enough eligible trajectories has no gradient estimate. We average gradients across trajectories and report the penalty-gradient ratio ρ from Equation 3. We also report the cosine between the two mean gradient directions and the task-gradient SNR, which compares the squared mean gradient norm with gradient variation across trajectories. Reversing both loss gradients leaves their cosine and norm ratio unchanged, but does not account for AdamW's momentum or coordinate-wise scaling. The measurements are taken before the update and do not change that update.

Table 3 averages the gradient diagnostics over valid iterations: three, two, three, and two at $\beta = 0, 0.1, 0.5, 1$, respectively. Its failure-persistence column uses each run's final training batch, including batches without a valid gradient estimate, so the columns summarize different portions of the same run. At $\beta = 1$, the final iteration has no gradient estimate; the reported ratio and cosine average the first two iterations. The ratio ρ exceeds one at $\beta = 0.5$ and $\beta = 1.0$, reaching 1.53 and 2.64, respectively. The iteration-averaged cosine is -0.145 at $\beta = 1.0$, combining values of 0.171 and -0.462. The task-gradient SNR values are 0.106, 0.076, 0.109, and 0.172, so they do not decrease monotonically with β. Only two to six trajectories have nonzero, varying GRPO advantages in these measurements. The limited sample size and the changes in trained policies can both affect the estimates. Thus, increasing ρ shows a larger relative sampled-penalty gradient in the measured parameters; it does not establish decreasing full-model SNR. For fixed sampled tokens, the penalty gradient is independent of the reference probabilities, so these measurements are not independent evidence for a reference-dependent KL mechanism; Appendix D.1 details the gradient derivation and numerical check. The gradient and observation-swap experiments also use different trajectories, so they do not trace a single causal path from an update to a later failure.

D.3 Distance from the Initial Parameters

Table 27 measures the Euclidean distance between trained parameters θ and initial parameters θ_0.

At $\beta = 1.0$, the distance is 13.40, about 59% of the distance 22.84 at $\beta = 0$. The parameters therefore move from their initial state even at the high coefficient. The error-type percentages in Figure 3 come from the separate 1.7B experiment and are reported in Appendix E.2.

Table 27: Parameter distance from initialization in Qwen3-0.6B BFCL runs.

β	$\|\theta - \theta_0\|$
0.0	22.84
0.01	23.42
1.0	13.40

E Feedback, Action Errors, and Input Dependence

Appendix E.1 describes observation replacement and response-change scoring. Appendix E.2 lists the action-error frequencies shown in the main text. Appendix E.3 defines the distinction between output diversity and input dependence. Appendix E.4 measures action diversity and its relation to parameter changes. Appendix E.5 defines the prompt-retrieval diagnostic; Appendix E.6 reports its results.

E.1 Observation-Swap Protocol

We measure response sensitivity by replacing only the latest observation and comparing the resulting response with the original one.

The test uses the Qwen3-4B base model and seed-42 GRPO checkpoints on 15 HotPotQA evaluation tasks per condition. At each noninitial assistant turn, we replace the latest user observation with a user message from another task's rollout. The donor message uses the corresponding turn index, or the donor's last available user message if the index is too large. The rest of the recipient history stays unchanged. Each eligible turn uses three different donor tasks. We request temperature 0.01 and a 256-token limit for generation with both the original and replacement observations. The saved records do not contain the full effective decoding configuration or random-number state.

The change detector compares lowercased response text after trimming whitespace. Identical responses count as unchanged. For different responses with word sets W_1 and W_2, it counts a change when $|W_1 \cap W_2| / \max(|W_1|, |W_2|) < 0.8$. Different responses with an empty word set also count as changed. This compares words, not parsed tool calls or semantic equivalence. The detector uses the full response, but the saved comparison records retain only its first 100 characters and the resulting change flag. The saved prefixes therefore do not allow an independent recomputation of every change decision. We average the three swaps within each turn, then average over turns for Table 4. The base model and four trained conditions contribute 243, 297, 282, 315, and 315 comparisons. Comparisons from the same task are not independent samples.

The measured sensitivity is lower for all tested trained policies than for the base model. The test does not establish zero mutual information, state-independent errors, or a monotonic effect of β. It lacks an unchanged-observation regeneration control, and a donor observation may be unreachable from the recipient's history. Different policies also supply different histories. These limits prevent a direct causal comparison of recovery in matched states.

E.2 The Recorded Error Breakdown

Figure 3 separates missing calls, wrong functions, and wrong arguments. The missing-call rate is 10.0% at $\beta = 0$ and 6.9% at $\beta = 0.1$. The latter condition has a correct-call rate of 13.5%. At $\beta = 0.5$, wrong arguments account for 29.2%, compared with 23.5% at $\beta = 0.1$. At $\beta = 1.0$, missing calls account for 89.2%. These categories describe what went wrong, but do not directly measure whether the model used the current observation. Testing whether argument choice loses observation sensitivity before function choice would require separate measurements of both.

E.3 Diversity and Input Dependence

For the input-dependence discussion, let X denote the input history and Z the generated output. Conditional entropy $H(Z \mid X)$ measures output randomness given the input, while mutual information $I(X; Z)$ measures dependence between input and output.

Definition E.1 (Template collapse (Wang et al., 2026)). *Template collapse means that $H(Z \mid X)$ remains high while $I(X; Z)$ approaches zero.*

Output diversity and dependence on the input are different properties. A model can produce many different responses while ignoring information needed for the task. Definition E.1 describes this case using conditional entropy and mutual information. Conversely, repeated short outputs may have low diversity without being fully independent of the input. An entropy bonus encourages variation in outputs (Haarnoja et al., 2018), but does not by itself ensure that the variation helps solve the task.

E.4 Action Diversity

The original analysis also records how varied the generated action sequences are. It converts outputs into action types, counts consecutive groups of three types, and computes the entropy of their pooled frequencies in bits. A sequence shorter than three contributes its full sequence as one group. We denote this action-group entropy by $\mathcal{H}_{\text{traj}}$. It measures diversity of these action groups, not dependence on the task input.

We also measure the mean squared parameter change from the base model. We call this quantity d_{param} to distinguish it from the policy KL divergence D_{KL}:

$$d_{\text{param}} = \frac{1}{N_\theta} \|\theta - \theta_0\|_2^2,$$

where N_θ is the number of model parameters. Table 28 reports correlations involving this parameter-distance measure and the action-diversity trends across β. We use r_{corr} for a correlation coefficient so that it cannot be confused with the penalty-gradient ratio ρ.

Table 28: Parameter-distance correlations and action-diversity patterns in the Qwen3-0.6B analysis. An asterisk marks a reported $p < 0.05$.

Domain	$r_{\text{corr}}(\beta, d_{\text{param}})$	p	Action diversity as β increases
FC	-0.30	.188	decreases
Math	-0.64*	.014	increases
Code	-0.56*	.038	no consistent direction
Web	-0.49	.077	increases

Action diversity falls with β in FC but rises in Math and Web in this summary. The differing directions do not identify the reason for a change or establish a relation between policy KL and input dependence.

E.5 Retrieval-Score Calculation

The retrieval score asks whether a generated completion can be matched to its original prompt. For completion z from prompt x_i, let $\ell_j(z)$ be its mean token log-likelihood under candidate prompt x_j. Retrieval accuracy is the fraction of completions for which the original prompt has the largest score. The experiment uses 16 candidate prompts and four completions per prompt, so random selection gives accuracy $1/16$. We also report the *score gap*, the original-prompt score minus the log of the mean exponentiated score over the $M = 16$ candidates:

$$\text{gap}(z) = \ell_i(z) - \log\left(\frac{1}{M}\sum_{j=1}^{M} \exp \ell_j(z)\right).$$

The scores are normalized by completion length, so this gap is not a calibrated estimate of sequence-level mutual information.

At $\beta = 1.0$, the matched score is -0.024, compared with about -0.07 in other conditions. The gap standard deviation is 0.184 rather than about 0.3, and generation takes 116 seconds rather than about 300. These observations do not show that output length causes the perfect retrieval score. Small prompt-specific wording or formatting differences can support correct ranking without useful tool

calls. The practical result is that this retrieval score does not flag the observed missing-call failure. Other ways of estimating input dependence require their own evaluation.

E.6 Does an input-dependence diagnostic detect missing tool calls?

We test the input-dependence diagnostic of Wang et al. (2026) on the missing-call condition identified in Section 6.2. Using the retrieval accuracy and score gap defined in Appendix E.5, we compare whether prompt identification distinguishes useful calls from missing calls. On Qwen3-1.7B BFCL outputs, we compare this accuracy across β. Retrieval accuracy is 1.0 at every tested β, including $\beta = 1$, where 89.2% of outputs contain no tool call (Table 29). Matching a response to its prompt rewards prompt-specific likelihood differences, whereas useful tool use requires choosing a task-appropriate action; this distinction explains how a perfect retrieval ranking can coexist with missing calls. Failure persistence and error types reveal the task-relevant differences that this particular diagnostic misses.

Table 29: Input-retrieval diagnostic results for Qwen3-1.7B on BFCL. Score gap reports the mean; SD is its standard deviation across completions, and time is in seconds.

β	Retrieval accuracy	Score gap	SD	Time (s)
0.0	1.000	2.237	0.273	269
0.001	1.000	2.248	0.265	329
0.01	1.000	1.640	0.321	344
0.05	1.000	1.591	0.363	335
0.1	1.000	2.227	0.294	298
0.5	1.000	1.971	0.305	316
1.0	1.000	2.698	0.184	116

F Training and Generation Controls

Appendix F.1 compares weight decay under supervised fine-tuning. Appendix F.2 reports thinking-mode and entropy-bonus results. Appendix F.3 examines the implemented coefficient schedule and its outcomes.

F.1 Supervised Fine-Tuning

Supervised fine-tuning (SFT) trains on fixed input–target examples rather than newly generated, reward-scored trajectories. The SFT control uses the same FC data, two seeds, and weight decay in $\{0, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. Task success remains near 0.91 and $\mathcal{H}_{\text{traj}}$ near 8.28. The reported rank correlation between d_{param} and $\mathcal{H}_{\text{traj}}$ is 0.309 with $p = 0.385$. This control does not resolve a best interior weight-decay value or a significant correlation. Since both the training objective and regularizer change relative to RL, it does not show that on-policy sampling is necessary for template collapse.

F.2 Thinking Mode

Figure 6 and Table 5 compare generation with and without reasoning text before the action.

For FC at $\beta = 0$, the score decreases from 0.819 to 0.404. At $\beta = 1.0$, it increases from 0.679 to 0.858 in FC and from 0.383 to 0.483 in Math. These comparisons measure task performance, not changes in mutual information or gradient SNR.

Entropy bonus. Table 30 varies λ_H, the weight of the entropy bonus that encourages less concentrated output probabilities.

At $\lambda_H = 0.01$, performance improves at $\beta = 0$ but worsens at $\beta = 0.01$. All four runs at $\lambda_H = 0.1$ are recorded as failed, without numerical scores or measurements that identify the failure mechanism. These results do not establish whether either entropy or input dependence changed as intended.

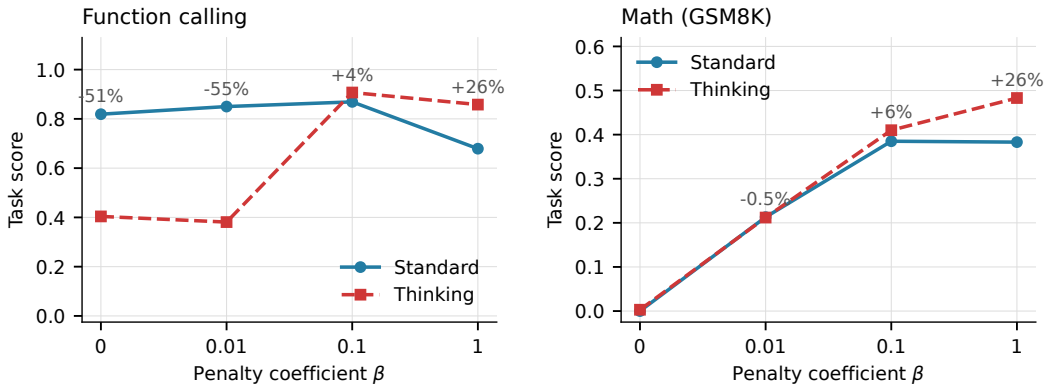

Figure 6: Task scores with and without pre-action reasoning for Qwen3-0.6B. Solid and dashed lines show standard and thinking-mode generation; annotations give relative score changes, not percentage-point differences.

Table 30: FC task scores for Qwen3-0.6B under different entropy coefficients. Dashes mark failed runs without reported numerical scores.

λ_H	β=0	β=0.01	β=0.1	β=1.0
0 (baseline)	0.819	0.850	0.869	0.679
0.01	0.906	0.693	0.879	0.615
0.1	—	—	—	—

F.3 Scheduling Ablation

We first derive the trajectory-level coefficient produced by averaging a proposed turn-level schedule, then compare its measured outcomes with fixed-coefficient training.

The implemented loss. We construct a list of turn-specific coefficients and average them before weighting the trajectory's sampled-penalty loss. For a coefficient that decreases linearly from 0.5 to 0, an episode of T turns therefore uses

$$\bar{\beta}(T) = \frac{1}{T}\sum_{t=0}^{T-1} 0.5\left(1 - \frac{t}{\max(T-1,1)}\right) = \begin{cases} 0.5, & T = 1, \\ 0.25, & T \geq 2. \end{cases}$$

No turn-specific weighting remains in the loss. We apply the same averaging rule on HotPotQA, WebShop, and InterCode. The comparisons below use an effective coefficient of 0.25 for the averaged schedule.

Recorded comparisons. The values below come from each run's last recorded batch of training trajectories, before that iteration's update. On HotPotQA with seed 42, fixed 0.5 versus the averaged schedule gives failure persistence $0.849 \to 0.861$, sequential consistency $0.680 \to 0.345$, and reward $0.314 \to 0.298$. For seed 123, the corresponding values are $0.881 \to 0.923$, $0.605 \to 0.450$, and $0.236 \to 0.199$. Neither comparison reduces failure persistence. On WebShop with seed 42, failure persistence changes from 0.200 to 0.775 and reward from 0.898 to 0.275. On InterCode with seed 42, the corresponding changes are $0.719 \to 0.871$ and $0.559 \to 0.396$.

The fixed-0.25 HotPotQA control gives failure persistence 0.849 in both seeds after rounding. Its coefficient matches the effective coefficient of the averaged schedule, yet the measured results differ. Without replaying matched runs, we cannot assign that difference to a specific source. These records therefore do not test whether turn-specific scheduling improves recovery. That test requires applying each coefficient to the intended turn's tokens and verifying the resulting loss weights.

G Scope of the Evidence and Further Tests

Appendix G.1 summarizes training duration, sample coverage, and algorithm coverage. Appendix G.2 states the comparisons needed to test causal error propagation and recovery.

G.1 Limits of the Current Evidence

To keep the conclusions proportional to the evidence, we summarize limits from training duration, seed coverage, evaluation samples, and algorithm coverage.

Training duration. The WebShop reward values over three recorded iterations are $0.709 \rightarrow 0.668 \rightarrow 0.985$ at $\beta = 0$ and $0.709 \rightarrow 0.580 \rightarrow 0.910$ at $\beta = 0.5$. At $\beta = 1.0$, reward decreases from 0.709 to 0.263 to 0.150, while failure persistence increases from 0.387 to 0.613 to 0.773. This documents deterioration over the observed window, not convergence or an inability to improve with further training. The initial BFCL runs last five iterations; most 4B and 8B runs last three. Longer training could change the coefficient ordering.

Seeds and evaluation samples. The audited HotPotQA and WebShop tables each have five reported runs per condition. The InterCode and ALFWorld extensions provide five-run mean/SE summaries, but the added per-seed records are unavailable. The BFCL transition analysis uses Qwen3-1.7B with seed 42, while the separate Qwen3-0.6B task-success analysis uses three seeds. Runs at different model sizes do not form repeated seeds of one configuration. Standard errors across training runs and episode-bootstrap intervals measure different sources of variation (Agarwal et al., 2021).

Algorithms and model families. Most evidence uses GRPO with Qwen. PPO covers three benchmarks with one seed, and Llama covers one HotPotQA experiment with one seed and SGD. These comparisons show that repeated failures occur in more than one setting, but their coefficient orderings differ. They do not establish one best coefficient across algorithms or model families.

G.2 What Remains to Be Tested

To show what would strengthen or weaken the proposed explanation, we state the missing tests and the limits of the current comparisons.

Testing the proposed explanation. A direct test would place different policies in the same reachable state after a controlled error, then compare their observation sensitivity and recovery. If sensitivity falls without any change in recovery in those matched states, that would weaken the proposed link between them. The current observation swaps and transition summaries do not provide this test.

Comparing tasks. Changing benchmarks changes more than the number of turns. Differences in task difficulty, labels, and available actions can also change which coefficient performs best. The results therefore do not support a general rule that longer tasks require a lower or higher β.

Interpreting all-pass rates. Equation 4 concerns episodes in which every scored step passes. It cannot replace a task-success measurement when later actions can repair earlier mistakes. With high per-turn error, its constant-error illustration is near zero for all conditions and is not useful for ranking them.

One scored step. When an episode contains only one scored step, its all-pass probability is simply $P(p_1)$. It has no adjacent-step transition and contributes nothing to either transition-rate denominator.

Recovery and consistency. Equation 1 shows why recovery and sequential consistency must be reported together. Neither rate alone determines task success, and neither directly measures input dependence.