DON'T SOLVE, JUST COMPARE: TINY ADVISORS FOR RUNTIME INTERVENTION IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents are emerging as an important paradigm for real-world tasks that require reasoning, tool use, and sequential decision-making. As these agents operate over longer horizons, runtime intervention offers a way to improve reliability without retraining the underlying actor. Effective intervention must provide a useful direction for recovery besides a warning. Existing approaches often rely on an expert solver or a critic that generates task-specific corrections, incurring either the cost of another capable solver or the capacity demands of a task-capable critic. We introduce **C**omparison-**O**nly **T**iny **A**dvisor (**COTA**) for constructive runtime intervention, which reduces the learned intervention role to local action comparison. A lightweight comparator judges the actor's proposal against available alternatives, and preferred alternatives are returned as non-binding advice for replanning. The comparator is trained from same-prefix counterfactual branches. Across WebShop, ALFWorld, and τ^3-Retail with three LLM actors, COTA instantiated with a 0.5B comparator consistently improves the original actor and achieves the strongest overall performance–cost trade-off among the compared methods. These results suggest that effective runtime intervention need not itself be a task-solving problem: the intervention role can be separated from task solving and handled by a lightweight model specialized for local comparison.

1 INTRODUCTION

Large language model (LLM) agents are increasingly used for long-horizon tasks that require sequential reasoning, tool use, and interaction with external environments (Yao et al., 2022b; Shinn et al., 2023). In such settings, decisions are consequential: a locally poor action can alter the environment, consume a limited interaction budget, and make subsequent recovery substantially harder. *Runtime intervention* provides a practical way to improve reliability without retraining or replacing the underlying actor, by monitoring an ongoing trajectory and redirecting execution when needed (Vasudev et al., 2026; Zhang et al., 2026; Jiang et al., 2026). Yet effective runtime intervention is not merely a failure-detection problem. Recent work shows that even accurate failure predictors can reduce end-to-end performance by disrupting trajectories that would otherwise succeed (Vasudev et al., 2026), and argues that effective runtime control should depend on whether an available intervention improves the downstream outcome, rather than on continuation risk alone (Zhang et al., 2026). Effective runtime intervention therefore requires more than detecting problematic decisions. It must also provide a recovery signal that improves what the agent does next.

Existing approaches pursue this goal through several routes. Self-reflection asks the actor to reconsider its own proposal, but without an external signal it can preserve or even amplify the same reasoning error (Shinn et al., 2023; Huang et al., 2024; Kamoi et al., 2024). Expert-handoff methods intervene by transferring control to a stronger solver when the current trajectory appears problematic (Zhang et al., 2026; Ong et al., 2025; Shen et al., 2024); while potentially effective, they require invoking another capable model. Critic-based methods instead keep the actor in control and use a separate model to generate task-specific corrective feedback (Jiang et al., 2026), but producing useful corrections still demands substantial task understanding from the critic, making it difficult to keep lightweight (Chen et al., 2024a; Ong et al., 2025). More broadly, reward-model-based inference-time search can improve agent performance by generating multiple complete trajectories and selecting the highest-scored one (Xia et al., 2025; Choudhury, 2025; Xi et al., 2026). Its effectiveness depends on reliable ranking, while larger search budgets require more trajectory rollouts and therefore higher inference cost. This

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.

motivates the question we study in this work: Can effective constructive runtime intervention be realized by a separate, lightweight intervention mechanism while requiring substantially weaker learned capabilities than task solving or correction generation?

Doing so presents two challenges. First, constructive intervention must provide more than a binary warning: when the actor proposes a poor action, it should offer a useful direction for replanning (Bai et al., 2022; Gou et al., 2023). Yet asking the tiny learned intervention module itself to discover that direction would reintroduce open-ended task solving (Lin et al., 2024; Chen et al., 2024b). Second, intervention should reflect the long-term consequence of the current decision rather than its surface plausibility (Sutton, 1988; Pignatelli et al., 2024; Xiong et al., 2024). Learning accurate absolute values for arbitrary state–action pairs is particularly demanding in long-horizon agent tasks (Arjona-Medina et al., 2019; Gehring et al., 2022; Parisi et al., 2022; Xi et al., 2026), while comparing actions observed under different trajectories mixes the effect of the action with differences in history and continuation policy (Oberst & Sontag, 2019; Jiang & Li, 2016; Thomas et al., 2015).

These challenges motivate a weaker primitive: comparison. An action need not be certified as globally good to determine that the actor's current proposal is locally weak. If the proposed action is repeatedly outperformed by plausible executable alternative actions, then the proposed action is likely a poor local choice, even when none of those alternative actions is itself optimal. Based on this observation, we propose **C**omparison-**O**nly **T**iny **A**dvisor (**COTA**), a comparison-only framework for constructive runtime intervention. First, to make comparison constructive, at each decision point, we obtain a small set of alternative actions from a reference candidate mechanism whose construction is left configurable, and a tiny comparator performs only one primitive: predicting which of two actions from the same state leads to the better continuation. Repeated pairwise judgments estimate how frequently the actor's proposal is dominated by its alternatives. When this candidate-relative domination is sufficiently strong, the winning alternatives are returned as non-binding advice and the actor replans before execution. Second, to learn this long-horizon comparison without absolute value estimation or cross-trajectory confounding, we use same-prefix counterfactual branching, inspired by Zhang et al. (2026). Starting from an identical environment state, we vary only the branch-point action and then return control to the same frozen actor. The resulting sibling trajectories provide pairwise supervision that directly matches the comparison required at runtime. The comparator therefore neither generates corrective actions nor estimates absolute action values. It only compares.

We evaluate COTA on WebShop (Yao et al., 2022a), ALFWorld (Shridhar et al., 2020), and τ^3-Retail (Yao et al., 2024), using Qwen3-8B, Qwen3.6-35B-A3B, and DeepSeek-V4-Flash as LLM agent actors. Across all evaluation combinations, despite using only a 0.5B comparator, COTA consistently improves the original actor while maintaining a favorable performance–cost trade-off. The gains extend to substantially stronger actors, showing that effective constructive intervention can be separated from task solving and realized by a lightweight model specialized for local comparison, while leaving final replanning to the actor. Our contributions are threefold:

- We formulate *constructive runtime intervention* and show that the capability required for its intervention role can be reduced from open-ended task solving or correction generation by a strong LLM to candidate-relative comparison, which can be handled by a separate, lightweight model.
- We propose COTA, a comparison-only framework that aggregates pairwise judgments through a statistically interpretable Monte Carlo gate. We further construct reliable pairwise supervision from same-prefix counterfactual trajectories, directly matching the comparison required at runtime.
- We evaluate COTA across three interactive environments and three actor families. A 0.5B comparator improves all nine actor–environment combinations, transfers without retraining to substantially stronger actors, and achieves a favorable performance–cost trade-off.

2 Problem Formulation

Agent trajectory. Following the standard formulation of interactive LLM agents (Yao et al., 2022b) and sequential decision making under partial observability (Kaelbling et al., 1998), we consider a frozen LLM agent π interacting with an environment to accomplish a task x. We refer to the frozen LLM policy π as the *actor*. At each step t, it observes o_t, produces a reasoning trace or plan p_t, and proposes an environment action a_t. If a_t is executed, the environment returns the next observation

o_{t+1}. We denote the trajectory prefix before executing a_t as:

$$\tau_t = (o_0, p_0, a_0, \ldots, o_{t-1}, p_{t-1}, a_{t-1}, o_t, p_t), \tag{1}$$

and let $s_t = (x, \tau_t)$ denote the actor-visible decision state. Thus, the proposed action is sampled as $a_t \sim \pi(\cdot \mid s_t)$. Here s_t denotes the information available to the agent at decision time, rather than necessarily full environment state.

Runtime intervention. In this paper, we consider runtime intervention (Wang et al., 2026a; Chen et al., 2025; Wang et al., 2026b) at the *pre-execution* stage. The agent has already proposed a_t, but the action has not yet been sent to the environment. We follow the general human-intervention and shielding paradigm that an external mechanism monitors a proposed action and may prevent or modify it before execution (Saunders et al., 2018; Alshiekh et al., 2018). Let $g_t \in \{0, 1\}$ denote the intervention decision. The original proposal is executed if $g_t = 0$ and withholds if $g_t = 1$. Our definition does not specify what happens after an action is withheld.

For any action a available at s_t, we define its action value (Watkins & Dayan, 1992; Sutton & Barto, 2018) under the actor π as:

$$Q^\pi(s_t, a) = \mathbb{E}\left[R(\tau) \mid s_t, ; a_t = a, ; \pi \text{ thereafter}\right], \tag{2}$$

where $R(\tau)$ is the return of the completed trajectory. $Q^\pi(s_t, a)$ is the expected return from taking a at s_t and following π thereafter.

Constructive runtime intervention. Compared with prior runtime monitoring and intervention methods (Xiang et al., 2025; Wang et al., 2026b), which primarily determine whether a proposed action a_t should be allowed to proceed or interrupted, without necessarily providing a concrete direction for recovery, we emphasize that a more constructive runtime intervention should provide an advisory action a_t^{adv} to the actor π when withholding a_t, thereby guiding subsequent action selection. The advice is non-binding: the actor replans after receiving it and produces a new proposal a_t'. We call the intervention constructive when the advice offers a better direction:

$$Q^\pi(s_t, a_t^{\text{adv}}) > Q^\pi(s_t, a_t).$$

The desired end-to-end outcome is an improved proposal:

$$Q^\pi(s_t, a_t') > Q^\pi(s_t, a_t).$$

We call the separate role that performs this advice-producing intervention an *advisor*. Our goal is to enable such intervention with a lightweight advisor that need not solve the task independently.

3 METHODOLOGY

In this section, we present COTA in four parts. We first introduce the comparison intuition underlying our approach in Section 3.1, and then describe the COTA runtime intervention pipeline in Section 3.2. We next describe the design of comparator training using same-prefix branches in Section 3.3, and finally analyze the statistical accuracy of COTA's intervention gate in Section 3.4.

3.1 INTUITION

Directly estimating whether π's proposal action a_t is "good" through $Q^\pi(s_t, a_t)$ is unnecessarily difficult for constructive runtime intervention. It is sufficient to determine whether a_t is poor compared with plausible alternatives available at the same decision point. Let $\mu(\cdot \mid s_t)$ denote the distribution over executable actions induced by a reference candidate mechanism. We define the *candidate-relative domination rate*:

$$\rho_\mu(s_t, a_t) = \Pr_{A \sim \mu(\cdot \mid s_t)} \left(Q^\pi(s_t, A) > Q^\pi(s_t, a_t)\right). \tag{3}$$

This quantity has a simple rank interpretation. If $\rho_\mu(s_t, a_t) = 0.1$, only 10% of candidates sampled from μ outperform the proposal; if $\rho_\mu(s_t, a_t) = 0.8$, the proposal a_t is dominated by 80% of the candidate distribution μ. Equivalently, $1 - \rho_\mu$ gives a_t's quantile in μ, up to ties. Importantly, μ need not be an expert policy or contain the globally optimal action. If ρ_μ is small, a_t already compares favorably with ordinary alternatives; conversely, a large ρ_μ provides evidence that even ordinary alternatives frequently improve upon it. Moreover, the alternatives that defeat a_t naturally serve as advice for replanning. This motivates a much narrower learned primitive for constructive runtime intervention: predicting which of two actions from the same state leads to the better continuation.

3.2 COMPARISON-ONLY TINY ADVISOR

Based on this statistical insight, we propose COTA, a runtime intervention framework that realizes the advisor using a single learned primitive: pairwise action comparison. At step t, π first proposes $a_t \sim \pi(\cdot \mid s_t)$. We then obtain K executable alternative actions from μ, i.e., $\mathcal{A}_t = \{A_1, \ldots, A_K\}$, $A_i \sim \mu(\cdot \mid s_t)$. A tiny comparator C_θ judges each candidate A_i only relative to a_t. C_θ's target is:

$$C_\theta(s_t, a_A, a_B) \approx \mathbb{I}\left[Q^\pi(s_t, a_A) > Q^\pi(s_t, a_B)\right]. \tag{4}$$

For each A_i, we define:

$$\widetilde{B}_i = C_\theta(s_t, A_i, a_t) \in \{0, 1\}, \tag{5}$$

where $\widetilde{B}_i = 1$ indicates that A_i is predicted to outperform a_t. COTA rejects a_t when at least R of the K alternatives defeat it:

$$\sum_{i=1}^{K} \widetilde{B}_i \geq R. \tag{6}$$

Otherwise, a_t is executed unchanged. When the gate fires, the predicted winners are ranked using C_θ, and the highest-ranked candidate a_t^{adv} is returned to π as non-binding advice. π then replans from the same s_t conditioned on this advice and produces a new proposal a_t', which is reviewed before execution. The comparator C_θ is therefore the only learned primitive required by COTA: it neither generates corrective actions nor estimates absolute action values, but only performs the local comparison in Eq. 4. π remains responsible for planning. COTA only assumes access to such a μ and is agnostic to how μ is constructed. We describe the concrete μ mechanisms used in our experiments in Section 4.2. We detail the statistical accuracy of this candidate-relative gate in Sec. 3.4.

3.3 COMPARATOR TRAINING

Training C_θ requires supervision for the relative continuation quality of two actions taken from the same state. Returns from unrelated trajectories are unsuitable for this purpose because they conflate the effect of the action with differences in preceding histories and subsequent behavior. We therefore construct training data using *same-prefix counterfactual branches*, following the classical idea of Monte-Carlo rollout evaluation from a shared decision state (Tesauro & Galperin, 1996; Bertsekas & Castanon, 1999).

At a sampled decision state s_t, we restore the same environment state, execute a branch-point action a, and then return control to the same frozen actor π. Let $Y(s_t, a) = R(\tau)$, $\tau \sim P(\cdot \mid s_t, a, \pi)$ denote the resulting branch return. By the definition of $Q^\pi(s_t, a)$ in Eq. 2:

$$\mathbb{E}[Y(s_t, a)] = Q^\pi(s_t, a). \tag{7}$$

Hence, $\widehat{Q}_M(s_t, a) = \frac{1}{M}\sum_{m=1}^{M} Y_m(s_t, a)$ is a Monte Carlo estimate of $Q^\pi(s_t, a)$ with M independent continuations. For sibling actions a_A and a_B, their empirical branch returns directly provide pairwise supervision:

$$\widehat{B}(s_t, a_A, a_B) = \mathbb{I}\left[\widehat{Q}_M(s_t, a_A) > \widehat{Q}_M(s_t, a_B)\right]. \tag{8}$$

We fine-tune C_θ to predict these pairwise labels from the actor-visible state and the two candidate actions. Because both branches share the same prefix and continuation actor, the comparison isolates the downstream consequence of changing the branch-point action. Details of label construction and comparator implementation are provided in Section 4.2.

3.4 STATISTICAL ACCURACY

In this part, we analyze the statistical accuracy of COTA's candidate-relative gate defined in Sec.3.2. Suppose first that the true pairwise relation were observable, and define $B_i = \mathbb{I}\left[Q^\pi(s_t, A_i) > Q^\pi(s_t, a_t)\right]$. For K sampled candidates, let:

$$\widehat{\rho}_K = \frac{1}{K}\sum_{i=1}^{K} B_i, \qquad \mathbb{E}[\widehat{\rho}_K] = \rho_\mu(s_t, a_t). \tag{9}$$

Thus, $\widehat{\rho}_K$ is a Monte Carlo estimate of the candidate-relative domination rate $\rho_\mu(s_t, a_t)$ in Eq. 3. Further, requiring R winners in Eq. 6 corresponds to testing whether $\rho_\mu(s_t, a_t) \geq \frac{R}{K}$, using $\widehat{\rho}_K$ as its Monte Carlo estimate. That is, whether a_t is dominated by at least an $\frac{R}{K}$ fraction of μ.

At deployment, we replace the oracle comparisons B_i with the learned predictions $\widetilde{B}_i$:

$$\widehat{\rho}_{\theta,K} = \frac{1}{K}\sum_{i=1}^{K} \widetilde{B}_i, \qquad g_\theta = \mathbb{I}\left[\widehat{\rho}_{\theta,K} \geq \frac{R}{K}\right]. \tag{10}$$

Let $\epsilon_\theta(s_t, a_t) = \Pr_{A\sim\mu(\cdot|s_t)}\left(C_\theta(s_t, A, a_t) \neq \mathbb{I}\left[Q^\pi(s_t, A) > Q^\pi(s_t, a_t)\right]\right)$ denote C_θ's error under μ. The estimation error then consists of comparator error and finite-candidate sampling error.

Proposition 3.1 (Estimation accuracy of the candidate-relative gate)**.** *For any fixed (s_t, a_t), assuming that the K candidates are sampled either independently from $\mu(\cdot \mid s_t)$, or uniformly without replacement from a finite candidate pool whose empirical distribution defines $\mu(\cdot \mid s_t)$:*

$$|\mathbb{E}[\widehat{\rho}_{\theta,K}] - \rho_\mu(s_t, a_t)| \leq \epsilon_\theta(s_t, a_t). \tag{11}$$

Moreover, with probability at least $1-\delta$, we have:

$$|\widehat{\rho}_{\theta,K} - \rho_\mu(s_t, a_t)| \leq \epsilon_\theta(s_t, a_t) + \sqrt{\frac{\log(2/\delta)}{2K}}. \tag{12}$$

The first term captures errors made by C_θ, while the second is the standard Monte Carlo error from using only K candidates. Consequently, when the true domination rate $\rho_\mu(s_t, a_t)$ is sufficiently far from the threshold $\frac{R}{K}$, the learned gate $g_\theta = \mathbb{I}[\widehat{\rho}_{\theta,K} \geq R/K]$ makes the same intervention decision as the oracle candidate-relative gate $\mathbb{I}[\rho_\mu(s_t, a_t) \geq R/K]$ with probability at least $1-\delta$. We provide the detailed proof and analysis in Appendix A. When intervention occurs, the same comparisons that establish that a_t is weak also identify alternatives predicted to have higher continuation value. COTA therefore obtains both components of constructive runtime intervention—*when* to interrupt and *what* direction to expose—from the same comparison primitive.

4 Experiments

4.1 Setups

Models. All agents in our experiments follow a ReAct-style interaction loop that alternates a textual rationale, one environment action, and the resulting observation. We evaluate Qwen3-8B [1] (Yang et al., 2025a), Qwen3.6-35B-A3B [2] (Qwen Team, 2026), and DeepSeek-V4-Flash (DeepSeek-AI, 2026). Unless stated otherwise, the tiny comparator in COTA is a full-parameter fine-tuned Qwen2.5-0.5B-Instruct [3] (Yang et al., 2024) comparator. The actor itself remains frozen in every condition.

Benchmarks. We study three environments with complementary action interfaces. (1) WebShop requires multi-step product search and purchasing under natural-language constraints, and provides partial reward in $[0, 1]$ (Yao et al., 2022a). We collect branch supervision from tasks 0–4999 and evaluate on the disjoint tasks 5000–5499. (2) ALFWorld evaluates compositional household tasks in a text-based embodied environment (Shridhar et al., 2020); we use its standard training split for data collection and the valid-unseen split for evaluation. (3) τ^3-Retail tests policy-constrained customer-service dialogue with state-changing tools and a simulated user (Yao et al., 2024); we retain the benchmark's official train/test partition. Splits are always made at the task level.

Metrics. WebShop uses mean reward in $[0, 1]$ as its primary metric, while ALFWorld and τ^3-Retail use success rate. For ease of comparison, all tables in the main content report performance on a common 0–100 scale: WebShop mean reward is multiplied by 100, while ALFWorld and τ^3-Retail success rates are reported in percentage points. This rescaling is purely for presentation and does not change the underlying evaluation metrics. Qwen actors are evaluated on 500 WebShop tasks

[1] https://huggingface.co/Qwen/Qwen3-8B

[2] https://huggingface.co/Qwen/Qwen3.6-35B-A3B

[3] https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct

and 134 ALFWorld valid-unseen games. Retail uses all 40 test tasks with three seeds, giving 120 episodes per system. To control cost, DeepSeek-V4-Flash uses fixed subsets of the same held-out WebShop and ALFWorld tasks, while retaining the full three-seed Retail protocol. We additionally report estimated token-based inference cost, computed from recorded input and output token counts using OpenRouter public list prices as of August 2026. Detailed pricing and cost accounting are provided in Appendix B.8.2.

Baselines. We compare COTA against three representative baselines. (1) *Self-Reflection* lets the actor reconsider and revise its own proposal without external feedback (Shinn et al., 2023). (2) *Asym-AC* represents critic-based runtime intervention, where a separate Qwen3-8B critic provides free-form corrective feedback before the actor replans (Jiang et al., 2026). (3) *AgentRM* represents reward-model-based inference-time search: it performs beam search over multiple candidate trajectories and selects outputs according to the learned reward model (Xia et al., 2025). Prompt templates in all benchmarks are provided in Appendix D.4. We detail the baseline selection and implementations in Appendix B.7.

4.2 Implementation Details

Prefix-branch supervision. COTA constructs comparator supervision with respect to a chosen frozen source actor. In our experiments, we use Qwen3-8B as this source actor for all three environments. We first complete a base trajectory, then restore selected prefixes and vary only the first branch action before handing control back to the same continuation actor. Branches are one level deep and are labeled solely by downstream environment outcomes. The branch sampling policy, and offline generation cost are deferred to Appendix B.2 and Appendix B.8.1.

Comparator implementation. Each example contains the actor-visible state and an ordered action pair, with labels `A`, `B`, or `T` denoting preference for the first action, preference for the second, or a tie. Each physical action pair is included in both A/B orders. After preprocessing, this yields 55k, 23k, and 9k supervision examples for WebShop, ALFWorld, and τ^3-Retail, respectively. At deployment, COTA accepts a winner only when the predictions under both input orders are semantically consistent. Full label construction and training details are provided in Appendix B.4.

Intervention hyperparameters. K is the number of alternatives provided each time, and R is the minimum number that must consistently beat the proposal a_t before intervention. We choose K and R by grid search on validation tasks using end-to-end task performance. This yields $K{=}4$, $R{=}1$ on WebShop and ALFWorld and $K{=}4$, $R{=}2$ on τ^3-Retail. Details are provided in Appendix B.5.

Reference candidate mechanisms. We instantiate μ in three ways: sampling legal environment actions, retrieving existing actions from offline data, or generating alternative actions with a 0.5B small language model. WebShop uses sampled environment actions; ALFWorld uses sampled environment actions, augmented with small-LM generation for stronger actors; and τ^3-Retail combines small-LM generation with offline retrieval. The comparator and intervention rule are unchanged across these configurations. Full construction details are provided in Appendix B.6.

4.3 Main Results

Performance. Table 1 shows that COTA consistently improves the original actor and achieves either the best or second-best performance in every setting. For Qwen3-8B, performance increases from 0.396 to 0.563 on WebShop, from 82.8% to 90.3% on ALFWorld, and from 37.5% to 45.0% on τ^3-Retail. The gains persist for the substantially stronger Qwen3.6-35B-A3B and DeepSeek-V4-Flash. This is important because the same benchmark-specific 0.5B comparator is not merely compensating for a weak actor: it remains useful even when the proposal is produced by substantially stronger models. More notably, each benchmark-specific comparator is trained only from Qwen3-8B branch supervision and reused unchanged for the stronger actors. Our cross-actor analysis in Appendix C.3 shows that source-actor preferences predominantly attenuate to ties rather than reverse direction under stronger continuation actors, providing a possible explanation for this transfer.

COTA outperforms both Self-Reflection and Asym-AC in all nine settings. AgentRM provides a substantially stronger comparison: its beam-search inference matches or exceeds COTA in several settings, particularly on τ^3-Retail and with DeepSeek-V4-Flash. However, this additional performance

Table 1: Comparison with baselines. Performance (Perf.) is mean reward on WebShop and success rate on ALFWorld and τ^3-Retail. Best performance is **bolded** and the second-highest distinct performance is underlined; the highest inference cost in each actor–environment setting is shown in red. Cost is the estimated API-equivalent inference cost in USD per 100 tasks. Avg. T is the mean task time normalized by the corresponding Original actor.

		WebShop			ALFWorld			τ^3-Retail		
Actor	Method	Perf.↑	Cost↓	Avg. T↓	Perf.↑	Cost↓	Avg. T↓	Perf.↑	Cost↓	Avg. T↓
Qwen3-8B	Original	39.6	\$0.698	1.00×	82.8	\$0.274	1.00×	37.5	\$2.00	1.00×
	Self-Reflection	35.3	\$1.54	2.45×	83.6	\$0.630	1.03×	39.2	\$4.42	1.66×
	Asym-AC	10.1	\$1.15	2.27×	82.8	\$0.492	1.03×	39.2	\$3.20	2.30×
	AgentRM	40.1	\$12.2	16.8×	85.1	\$3.61	20.6×	**50.8**	\$32.8	21.6×
	COTA (ours)	**56.3**	\$0.883	1.41×	**90.3**	\$0.378	1.14×	45.0	\$2.19	2.04×
Qwen3.6-35B-A3B	Original	56.6	\$0.362	1.00×	85.1	\$0.218	1.00×	62.5	\$0.613	1.00×
	Self-Reflection	58.1	\$0.736	2.00×	85.1	\$0.475	1.57×	62.5	\$1.55	2.08×
	Asym-AC	41.6	\$0.666	2.22×	85.8	\$0.438	1.43×	60.8	\$0.911	1.42×
	AgentRM	62.4	\$5.83	9.19×	87.3	\$3.23	9.22×	**68.8**	\$12.0	9.65×
	COTA (ours)	**68.1**	\$0.636	1.45×	**94.0**	\$0.231	1.30×	65.0	\$0.670	1.78×
DeepSeek-V4-Flash	Original	60.9	\$8.51	1.00×	90.0	\$0.133	1.00×	80.8	\$0.825	1.00×
	Self-Reflection	78.0	\$6.99	0.73×	95.0	\$0.204	0.91×	77.5	\$1.40	1.13×
	Asym-AC	70.3	\$6.26	0.58×	87.5	\$0.268	0.94×	80.0	\$0.897	0.79×
	AgentRM	**81.7**	\$76.4	8.81×	**100**	\$0.645	8.71×	**87.5**	\$8.45	11.0×
	COTA (ours)	79.6	\$3.19	0.49×	**100**	\$0.083	0.76×	82.5	\$0.882	1.17×

comes from substantially more inference-time search. Overall, COTA provides consistent gains across actor scales while remaining competitive with much more expensive search-based inference.

Inference cost. COTA achieves these gains with relatively small additional token cost. Its API-equivalent cost is higher than the Original actor in most settings, as expected from the "replanning design", but remains lower than both Self-Reflection and Asym-AC in all nine settings while simultaneously achieving higher performance. The contrast with AgentRM is larger: AgentRM costs 7.8–24.0× more than COTA across the nine settings. Thus, although search can occasionally attain higher performance, COTA provides a substantially cheaper way to improve the actor without requiring a large inference-time search budget.

Runtime. COTA incurs modest online overhead, averaging 1.28× the end-to-end episode time of the original actor, with seven of nine settings below 1.5×. The larger overheads occur on τ^3-Retail, where the chosen reference candidate mechanism additionally invokes a small LLM. By comparison, AgentRM requires 5.4–18.1× as much episode time as COTA because of its multi-trajectory beam search. Avg. T can occasionally fall below 1× because useful intervention may reduce the number of interaction steps required to complete the task.

4.4 Ablation Study

Holding μ fixed, we ablate two design choices in COTA: the *learning objective*, comparing absolute $Q^\pi(s, a)$ estimation with our pairwise comparator C_θ, and the *intervention mechanism*, comparing forced action execution with constructive runtime intervention as defined in Section 2. This yields a 2×2 ablation, which we evaluate with Qwen3-8B and Qwen3.6-35B-A3B. In this ablation, the absolute-Q estimation model is trained by fine-tuning the same Qwen2.5-0.5B backbone.

Table 2 shows that both design choices are important. First, directly executing the selected alternative action is consistently inferior to constructive intervention. Under absolute-Q supervision, replacing forced execution with constructive intervention raises Qwen3-8B from 2.24% to 57.46% on ALFWorld and from 4.17% to 16.67% on τ^3-Retail, with the same trend for Qwen3.6. The pattern remains under pairwise supervision: forcing the predicted winner is substantially worse than returning it as advice and allowing the actor to replan, particularly on the two long-horizon environments. These results support the formulation in Section 2: a useful advisor should guide the stronger actor rather than directly take over its next action.

Constructive intervention alone, however, does not account for the full gain. Holding μ fixed, replacing absolute-Q estimation with same-prefix pairwise comparison improves ALFWorld from 57.46% to 90.30% for Qwen3-8B and from 63.43% to 94.03% for Qwen3.6; on τ^3-Retail, the

Table 2: Ablation of learning objective and intervention mechanism. WebShop reports mean reward; ALFWorld and τ^3-Retail report success rate. "Constructive" denotes constructive runtime intervention: the proposed action a_t is withheld only when intervention is triggered, a preferred alternative is returned as advice, and the actor π replans before execution.

		WebShop		ALFWorld		τ^3-Retail	
Objective	Intervention	Qwen3	Qwen3.6	Qwen3	Qwen3.6	Qwen3	Qwen3.6
Absolute Q	Forced	31.5	41.9	2.24	8.96	4.17	5.00
Absolute Q	Constructive	54.9	64.5	57.5	63.4	16.7	17.5
Pairwise comparison	Forced	37.8	60.4	51.5	50.8	16.7	37.5
Pairwise comparison	Constructive	**56.3**	**68.1**	**90.3**	**94.0**	**45.0**	**65.0**

corresponding gains are $16.67\% \rightarrow 45.00\%$ and $17.50\% \rightarrow 65.00\%$. The pairwise objective directly matches the judgment required within COTA —whether an available alternative is better than current a_t—without requiring the comparator to estimate an absolute long-horizon value. The best performance therefore requires both components of COTA: a narrow comparison objective and constructive runtime intervention that delegates final replanning to the actor.

4.5 Constructive Replanning

A central property of COTA is that intervention provides additional local directions without replacing the actor's own decision process. We first ask whether the reference candidate mechanism μ exposes actions that repeated actor sampling would otherwise miss. Across 300 prefixes, eight high-temperature actor samples produce only 2.28 distinct executable actions on average (28.5% unique-action utilization), with all eight samples collapsing to the same canonical action for 150 prefixes. In contrast, four reference candidates produce 3.71 distinct actions on average (92.7% utilization), showing that the reference mechanism substantially broadens the local action support available for intervention.

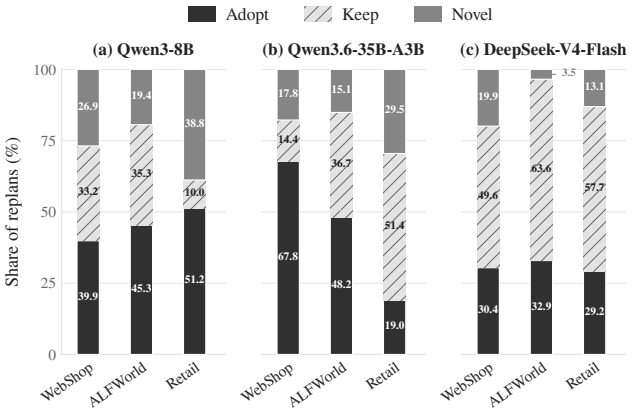


Figure 1: **Actor behavior after COTA intervention.** The actor's replan decision a'_t are categorized as *Adopt* if they adopt the advice a_t^{adv}, *Keep* if they repeat the original a_t, and *Novel* otherwise.

We then examine how this additional information affects replanning. Figure 1 categorizes the actor's next proposal as *Adopt*, *Keep*, or *Novel*. Actors frequently adopt recommended actions, but substantial mass remains on *Keep* and *Novel*, confirming that advice does not reduce replanning to forced selection. For example, DeepSeek-V4-Flash adopts only 29.2–32.9% of recommendations across the three environments while retaining its original proposal in 49.6–63.6% of replans. Qwen3.6 is more context-dependent, adopting 67.8% on WebShop but only 19.0% on τ^3-Retail. These results show that COTA can expose otherwise-missing local directions while preserving the actor's control over the final action.

4.6 Role of Repeated Intervention

Unlike one-shot correction, COTA remains active throughout the trajectory and may intervene again after the actor replans. We therefore ask whether its gains come from a few isolated interventions

Table 3: Performance under different episode-level intervention budgets with Qwen3-8B as the actor. B is the maximum number of decision states at which COTA may intervene within an episode; "Full" removes this cap. The "> 2 interv." column reports the fraction of Full-COTA episodes with more than two interventions.

Environment	$B = 0$	$B = 1$	$B = 2$	Full	> 2 interv.
WebShop	36.6	40.6	44.1	**56.3**	68.0%
ALFWorld	82.8	83.6	79.9	**90.3**	37.3%
τ^3-Retail	44.2	43.3	44.2	**45.0**	19.2%

or from keeping the intervention loop active over the episode. To test this, we cap the number of decision states at which COTA may intervene by an episode-level budget B.

We use Qwen3-8B as the actor. As shown in Table 3, restricting COTA to one or two interventions leaves substantial performance unrealized on WebShop and ALFWorld. On WebShop, mean reward increases from 0.366 at $B = 0$ to 0.406 and 0.441 at $B = 1$ and $B = 2$, but reaches 0.563 with unrestricted intervention; correspondingly, 68.0% of Full-COTA episodes contain more than two interventions. ALFWorld shows the same overall pattern despite non-monotonic small-budget results: $B = 1$ and $B = 2$ reach only 83.58% and 79.85%, compared with 90.30% under Full-COTA. In contrast, τ^3-Retail is relatively insensitive to the budget, consistent with only 19.2% of episodes containing more than two interventions. These results suggest that COTA's benefit is not necessarily concentrated in a single decisive correction; when useful intervention opportunities recur, keeping the lightweight comparison loop active throughout the trajectory can be important.

5 Related Work

Runtime Intervention. Runtime intervention has roots in human intervention and shielding for sequential decision making, where an external mechanism blocks or modifies unsafe actions (Saunders et al., 2018; Alshiekh et al., 2018). Recent LLM-agent guardrails extend this idea through policy enforcement and proactive risk monitoring (Chen et al., 2025; Wang et al., 2026a;b). Beyond safety, Vasudev et al. (2026) show that accurate failure prediction does not necessarily yield beneficial intervention, while Zhang et al. (2026) argue that runtime control should account for the downstream advantage of available interventions. Building on this shift from detecting failures to evaluating intervention utility, we study whether the mechanism that determines and guides intervention can itself be made lightweight.

Lightweight Auxiliary Models for Strong Actors. A growing line of work shows that smaller specialized models can assist substantially stronger executors through verification, workflow optimization, or critique (Kirchner et al., 2024; Nie et al., 2025; Dong et al., 2025; Yang et al., 2025b). In agent settings, Jiang et al. (2026) use a weaker critic for runtime intervention, while Jin et al. (2026) show that useful revision guidance can improve a stronger frozen model without requiring the critic to solve the task end-to-end. Building on this separation of roles, we push the idea further by narrowing the capability required of the auxiliary model from task-level solving or generation to local pairwise comparison, making the intervention role viable for a truly lightweight model.

6 Conclusion

We study how to improve long-horizon LLM agents without further post-training the actor or relying on another strong but expensive LLM for runtime intervention. Our results show that the intervention role can be separated from task solving and specialized into a lightweight module: COTA trains a 0.5B comparator for local action comparison while leaving task-level replanning to the original actor. Across diverse environments and actors, this design consistently improves the underlying agent while maintaining a favorable performance–cost trade-off. More broadly, our results suggest a practical division of labor for agent systems: keep strong LLMs focused on solving the task, and delegate narrow intervention decisions to cheap, specialized models.

AI USE STATEMENT

Generative AI tools were used to assist with implementation of experimental code, as well as to identify and explore relevant literature. The research methodology, experimental design, analysis, and interpretation of results were conducted by the authors. All AI-assisted code, experimental outputs, and cited references were reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

REFERENCES

Mohammed Alshiekh, Roderick Bloem, Ruediger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, pp. 2669–2678, 2018.

Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. Rudder: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems*, volume 32, pp. 13544–13555, 2019.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Dimitri P. Bertsekas and David A. Castanon. Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5(1):89–108, 1999.

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

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, 2024b.

Zhaorun Chen, Mintong Kang, and Bo Li. Shieldagent: Shielding agents via verifiable safety policy reasoning. In *International Conference on Machine Learning*, 2025.

Sanjiban Choudhury. Process reward models for llm agents: Practical framework and directions. *arXiv preprint arXiv:2502.10325*, 2025.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Guanting Dong, Jiajie Jin, Xiaoxi Li, Yutao Zhu, Zhicheng Dou, and Ji-Rong Wen. RAG-critic: Leveraging automated critic-guided agentic workflow for retrieval augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3551–3578. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.179.

Clement Gehring, Masataro Asai, Rohan Chitnis, Tom Silver, Leslie Pack Kaelbling, Shirin Sohrabi, and Michael Katz. Reinforcement learning for classical planning: Viewing heuristics as dense reward generators. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, pp. 588–596, 2022. doi: 10.1609/icaps.v32i1.19846.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

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

Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 652–661, 2016.

Shuli Jiang, Zhaoyang Zhang, Yi Zhang, Shuo Yang, Wei Xia, and Stefano Soatto. Asymmetric actor-critic for multi-turn llm agents. *arXiv preprint arXiv:2604.00304*, 2026.

Can Jin, Jiakang Li, Rui Wu, Eddy Zhang, and Dimitris N. Metaxas. Weak critics make strong learners: On-policy critique distillation for scalable oversight. In *3rd AI for Math Workshop: Toward Self-Evolving Scientific Agents*, 2026.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998. doi: 10.1016/S0004-3702(98)00023-X.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024. doi: 10.1162/tacl_a_00713.

Jan Hendrik Kirchner, Yining Chen, Harri Edwards, Jan Leike, Nat McAleese, and Yuri Burda. Prover-verifier games improve legibility of llm outputs. *arXiv preprint arXiv:2407.13692*, 2024.

Zicheng Lin, Zhibin Gou, Tian Liang, Ruilin Luo, Haowei Liu, and Yujiu Yang. Criticbench: Benchmarking llms for critique-correct reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 1552–1587. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.91.

Fan Nie, Lan Feng, Haotian Ye, Weixin Liang, Pan Lu, Huaxiu Yao, Alexandre Alahi, and James Zou. Weak-for-strong: Training weak meta-agent to harness strong executors. In *Proceedings of the Second Conference on Language Modeling*, 2025.

Michael Oberst and David Sontag. Counterfactual off-policy evaluation with gumbel-max structural causal models. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 4881–4890, 2019.

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

Simone Parisi, Davide Tateo, Maximilian Hensel, Carlo D'Eramo, Jan Peters, and Joni Pajarinen. Long-term visitation value for deep exploration in sparse-reward reinforcement learning. *Algorithms*, 15(3):81, 2022. doi: 10.3390/a15030081.

Eduardo Pignatelli, Johan Ferret, Matthieu Geist, Thomas Mesnard, Hado van Hasselt, and Laura Toni. A survey of temporal credit assignment in deep reinforcement learning. *Transactions on Machine Learning Research*, 2024.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL `https://qwen.ai/blog?id=qwen3.6-35b-a3b`.

William Saunders, Girish Sastry, Andreas Stuhlmueller, and Owain Evans. Trial without error: Towards safe reinforcement learning via human intervention. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2067–2069, 2018.

Zejiang Shen, Hunter Lang, Bailin Wang, Yoon Kim, and David Sontag. Learning to decode collaboratively with multiple language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12974–12990, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.701.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3 (1):9–44, 1988.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2 edition, 2018.

Gerald Tesauro and Gregory R. Galperin. On-line policy improvement using monte-carlo search. In *Advances in Neural Information Processing Systems*, volume 9, 1996.

Philip S. Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High-confidence off-policy evaluation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015. doi: 10.1609/aaai.v29i1.9541.

Rakshith Vasudev, Melisa Russak, Dan Bikel, and Waseem Alshikh. Accurate failure prediction in agents does not imply effective failure prevention. *arXiv preprint arXiv:2602.03338*, 2026.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. In *Proceedings of the IEEE/ACM International Conference on Software Engineering*, 2026a. doi: 10.1145/3744916.3764546.

Haoyu Wang, Christopher M. Poskitt, Jiali Wei, and Jun Sun. ProbGuard: Proactive runtime monitoring for LLM agent safety via probabilistic prediction, 2026b. Accepted at ASE 2026.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992. doi: 10.1007/BF00992698.

Zhiheng Xi, Chenyang Liao, Guanyu Li, Yajie Yang, Wenxiang Chen, Zhihao Zhang, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, Wei Wu, Tao Ji, Tao Gui, Qi Zhang, and Xuanjing Huang. Agentprm: Process reward models for llm agents via step-wise promise and progress. In *Proceedings of the ACM Web Conference*, 2026.

Yu Xia, Jingru Fan, Weize Chen, Siyu Yan, Xin Cong, Zhong Zhang, Yaxi Lu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Agentrm: Enhancing agent generalization with reward modeling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19277–19290, 2025.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. Guardagent: Safeguard llm agents via knowledge-enabled reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 68316–68342. PMLR, 2025.

Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. Watch every step! llm agent learning via iterative step-level process refinement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 1556–1572. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.93.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

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

Ruihan Yang, Fanghua Ye, Jian Li, Siyu Yuan, Yikai Zhang, Zhaopeng Tu, Xiaolong Li, and Deqing Yang. The lighthouse of language: Enhancing llm agents via critique-guided improvement. In *Advances in Neural Information Processing Systems*, volume 38, 2025b. doi: 10.52202/085713-5491.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022b.

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

Chubin Zhang, Zhenglin Wan, Xingrui Yu, Jingxuan Wu, Qi Wen, Pengfei Zhou, Wangbo Zhao, and Ivor Tsang. Calibration is not control: Why llm-agent oversight needs intervention. *arXiv preprint arXiv:2606.21399*, 2026.

A Theoretical Details

This section provides additional analysis for the methodology in Section 3.

A.1 Same-Prefix Branch Analysis

Fix a decision state s_t, a branch-point action a, and the frozen continuation actor π. By construction, a same-prefix branch first restores the environment to s_t, executes a, and then returns control to the same actor π. Recall that:

$$Y(s_t, a) = R(\tau), \qquad \tau \sim P(\cdot \mid s_t, a, \pi). \tag{13}$$

Therefore,

$$\begin{aligned} \mathbb{E}[Y(s_t, a)] &= \mathbb{E}_{\tau \sim P(\cdot \mid s_t, a, \pi)}[R(\tau)] \\ &= Q^{\pi}(s_t, a), \end{aligned} \tag{14}$$

which gives Eq. 7 in the main text.

For M independent continuations,

$$\widehat{Q}_M(s_t, a) = \frac{1}{M} \sum_{m=1}^{M} Y_m(s_t, a), \tag{15}$$

and linearity of expectation gives:

$$\mathbb{E}[\widehat{Q}_M(s_t, a)] = Q^{\pi}(s_t, a). \tag{16}$$

Whenever the return is integrable, $\widehat{Q}_M(s_t, a)$ also converges almost surely to $Q^{\pi}(s_t, a)$ as $M \to \infty$.

For two sibling actions a_A and a_B, define their true and empirical continuation-value gaps as:

$$d(s_t, a_A, a_B) = Q^{\pi}(s_t, a_A) - Q^{\pi}(s_t, a_B), \tag{17}$$

$$\widehat{d}_M(s_t, a_A, a_B) = \widehat{Q}_M(s_t, a_A) - \widehat{Q}_M(s_t, a_B). \tag{18}$$

Then we have:

$$\mathbb{E}[\widehat{d}_M(s_t, a_A, a_B)] = d(s_t, a_A, a_B). \tag{19}$$

Thus the branch-return difference isolates the downstream consequence of changing the branch-point action while keeping both the preceding state and the continuation actor fixed. This is the quantity thresholded to construct the empirical supervision in Eq. 8.

It is useful to distinguish unbiased value estimation from finite-sample pairwise labeling. Although $\widehat{Q}_M(s_t, a)$ is unbiased for $Q^\pi(s_t, a)$, the thresholded label $\mathbb{I}[\widehat{d}_M > 0]$ need not be an unbiased estimator of $\mathbb{I}[d > 0]$. In particular, comparisons with small true value gaps are more susceptible to finite-rollout noise.

For example, suppose branch returns are normalized to $[0, 1]$, and suppose $\widehat{d}_M$ is formed from M independent paired branch-return differences, each lying in $[-1, 1]$. If $d > 0$, Hoeffding's inequality gives:

$$\Pr(\widehat{d}_M \le 0) \le \exp\left(-\frac{Md^2}{2}\right). \tag{20}$$

Hence increasing the number of branch continuations primarily improves the reliability of supervision for comparisons whose true continuation values are close.

A.2 Proof of Proposition 3.1

Fix (s_t, a_t) throughout this subsection. For a candidate $A \sim \mu(\cdot \mid s_t)$, define:

$$B(A) = \mathbb{I}\left[Q^\pi(s_t, A) > Q^\pi(s_t, a_t)\right], \tag{21}$$

$$\widetilde{B}(A) = C_\theta(s_t, A, a_t). \tag{22}$$

For compactness, write:

$$\rho = \rho_\mu(s_t, a_t) = \mathbb{E}_{A\sim\mu}[B(A)], \tag{23}$$

$$q_\theta = \mathbb{E}_{A\sim\mu}[\widetilde{B}(A)]. \tag{24}$$

The pairwise error defined in Section 3.4 can be written as:

$$\epsilon_\theta = \Pr_{A\sim\mu}\left(\widetilde{B}(A) \neq B(A)\right). \tag{25}$$

Because both $B(A)$ and $\widetilde{B}(A)$ are binary, we have:

$$\begin{aligned} |q_\theta - \rho| &= \left|\mathbb{E}[\widetilde{B}(A) - B(A)]\right| \\ &\le \mathbb{E}[|\widetilde{B}(A) - B(A)|] \\ &= \epsilon_\theta. \end{aligned} \tag{26}$$

This step does not require independence across candidates; it is purely a single-candidate comparison between the learned and oracle relations.

Now suppose first that $A_1, \ldots, A_K$ are independently sampled from $\mu(\cdot \mid s_t)$, and define:

$$\widetilde{B}_i = \widetilde{B}(A_i), \qquad \widehat{\rho}_{\theta,K} = \frac{1}{K}\sum_{i=1}^{K} \widetilde{B}_i. \tag{27}$$

By linearity of expectation, we have:

$$\mathbb{E}[\widehat{\rho}_{\theta,K}] = q_\theta. \tag{28}$$

Combining Eqs. equation 26 and equation 28 yields:

$$|\mathbb{E}[\widehat{\rho}_{\theta,K}] - \rho| \le \epsilon_\theta, \tag{29}$$

which proves Eq. 11.

For the finite-candidate component, for independent candidate sampling, the concentration result follows directly from Hoeffding's inequality. The same bound also holds when the K candidates are sampled uniformly without replacement from a finite candidate population: sampling without replacement is at least as concentrated as sampling with replacement for bounded finite-population averages. Therefore, under either sampling scheme, we have:

$$\Pr(|\widehat{\rho}_{\theta,K} - q_\theta| \ge \xi) \le 2\exp(-2K\xi^2). \tag{30}$$

With probability at least $1-\delta$,

$$|\widehat{\rho}_{\theta,K} - q_\theta| \leq \sqrt{\frac{\log(2/\delta)}{2K}}. \tag{31}$$

Using the triangle inequality together with Eq. 26, we have:

$$\begin{aligned} |\widehat{\rho}_{\theta,K} - \rho| &\leq |\widehat{\rho}_{\theta,K} - q_\theta| + |q_\theta - \rho| \\ &\leq \epsilon_\theta + \sqrt{\frac{\log(2/\delta)}{2K}}. \end{aligned} \tag{32}$$

This proves Eq. 12 under both sampling schemes.

A.3 Gate Decision Consistency

Proposition 3.1 implies that, with probability at least $1-\delta$:

$$|\widehat{\rho}_{\theta,K} - \rho_\mu(s_t, a_t)| \leq \epsilon_\theta(s_t, a_t) + \sqrt{\frac{\log(2/\delta)}{2K}}. \tag{33}$$

Therefore, whenever the true domination rate is separated from the intervention threshold by more than this error:

$$\left|\rho_\mu(s_t, a_t) - \frac{R}{K}\right| > \epsilon_\theta(s_t, a_t) + \sqrt{\frac{\log(2/\delta)}{2K}}, \tag{34}$$

the learned estimate and the true domination rate lie on the same side of the threshold with probability at least $1-\delta$. Consequently, we have:

$$\mathbb{I}\left[\widehat{\rho}_{\theta,K} \geq \frac{R}{K}\right] = \mathbb{I}\left[\rho_\mu(s_t, a_t) \geq \frac{R}{K}\right] \tag{35}$$

with probability at least $1-\delta$.

Thus, away from an uncertainty region around the intervention threshold, the learned gate agrees with the oracle candidate-relative gate. Lower comparator error narrows this region through ϵ_θ, while increasing K reduces the finite-candidate sampling term at the standard $K^{-1/2}$ rate.

A.4 Candidate Distribution

The candidate-relative formulation does not require the reference mechanism μ to be an expert policy. However, constructive advice is necessarily limited by candidate support: the framework cannot expose a genuinely better alternative if the sampled candidate set contains none.

Under independent sampling from μ, the probability that at least one of K candidates truly improves on the current proposal is:

$$\Pr[\exists i : Q^\pi(s_t, A_i) > Q^\pi(s_t, a_t)] = 1 - [1 - \rho_\mu(s_t, a_t)]^K. \tag{36}$$

For the finite-pool without-replacement setting, suppose exactly W of the N available candidates have higher continuation value than the proposal. Then:

$$\rho_N = \frac{W}{N}. \tag{37}$$

The probability that a uniformly sampled subset of size K contains at least one true winner is:

$$\Pr(\text{at least one true winner}) = 1 - \frac{\binom{N-W}{K}}{\binom{N}{K}}, \tag{38}$$

where the numerator is interpreted as zero when $K > N - W$.

These expressions describe a support ceiling rather than a quality assumption on μ. A weak reference distribution can still provide a meaningful candidate-relative comparison, but it will supply genuinely useful alternatives less frequently. Conversely, broader or stronger candidate support increases the opportunities for constructive replanning without changing the comparator's learning objective. Appendix C.4 provides a controlled empirical study of this support effect on WebShop.

B Experimental Details

B.1 Evaluation Protocols and Data Splits

Table 4: Evaluation slices. A repeat denotes one complete environment episode with a distinct seeded rollout. DeepSeek uses cost-controlled fixed subsets for WebShop and ALFWorld.

Environment	Local tasks	DeepSeek tasks	Local repeats	DeepSeek repeats	Max steps
WebShop	500	50	1	1	20
ALFWorld	134	20	1	2	30
τ^3-Retail	40	40	3	3	100

WebShop. The branch-data pool uses tasks 0–4999. The held-out evaluation set is tasks 5000–5499, which is never used for branching, preprocessing, checkpoint selection, or hyperparameter sweeps. The environment returns a reward in $[0, 1]$ based on matched product attributes. We call reward 1 an exact success and separately report the fraction of reward-0 trajectories.

ALFWorld. Data collection uses training games, while evaluation uses the 134 games in the standard valid-unseen split. A trajectory succeeds only when the environment's terminal goal condition is satisfied. The actor sees the task, the executed action–observation prefix, the current observation, and the current admissible action set.

τ^3-Retail. We use the official Retail test split of 40 tasks. Each system is evaluated under three rollout seeds, and success is averaged over all 120 episodes. The environment includes a simulated user, policy text, and a database-backed tool interface; one actor turn may be followed by a tool result or a user reply.

B.2 Prefix-Branch Collection

We generate supervision in two stages. First, the frozen Qwen3-8B source actor completes a linear base trajectory without intervention. Second, after termination, we select decision points across the trajectory and restore each exact prefix. Where the environment does not support snapshots, restoration resets the task and replays all preceding actions.

For a base trajectory with T actions, the default sampler chooses $\min(\lceil 0.5T \rceil, 10)$ temporally stratified decision points. At each verified point, we keep the base action and obtain up to five distinct legal alternatives. The branching action differs across siblings; every branch thereafter returns to the same frozen continuation actor, low-temperature decoding, remaining step budget, and terminal rule. Branch trajectories are never branched recursively. This produces linear rather than exponential data growth while preserving the same-prefix counterfactual interpretation.

B.3 Training Data Construction

Within each verified prefix group, we construct both base–branch and branch–branch pairs. WebShop orders actions by downstream partial reward, while ALFWorld and Retail use terminal success. Small outcome differences are treated as ties when they do not support a reliable preference. Ties are downsampled to approximately 20% of the training data so that abundant same-outcome siblings do not dominate learning. Every retained physical pair is direction-augmented into both (A, B) and (B, A) orders. All members of a task or interaction tree remain in one split.

Table 5: Comparator data split details.

Environment	Physical pairs	Train	Validation	Test
WebShop	35,117	55,856	7,078	7,300
ALFWorld	14,548	22,678	3,222	3,196
τ^3-Retail	5097	8,910	1284	1278

B.4 Comparator Training and Inference

Section 3.2 defines the ideal binary comparison target as $C_\theta(s_t, a_A, a_B) \approx \mathbb{I}[Q^\pi(s_t, a_A) > Q^\pi(s_t, a_B)]$ in Eq. 4. Here we describe how this oracle target comparator is realized in our implementation. Training uses same-prefix branch estimates to supervise the relative ordering of two actions, with an additional tie class for indistinguishable outcomes. At inference time, the resulting ternary model outputs are mapped back to the binary decision C_θ required by the runtime intervention rule.

Training supervision. As described in Section 3.3, for two sibling actions a_A and a_B taken from the same state, we estimate their actor-conditioned continuation values using same-prefix Monte Carlo branches $\widehat{Q}_M(s_t, a_A)$ and $\widehat{Q}_M(s_t, a_B)$. The binary relation introduced in Eq. 8 $\widehat{B}(s_t, a_A, a_B) = \mathbb{I}\left[\widehat{Q}_M(s_t, a_A) > \widehat{Q}_M(s_t, a_B)\right]$ is therefore the empirical counterpart of the strict ordering in Eq. 4. Since $\widehat{Q}_M(s_t, a)$ is a Monte Carlo estimate of $Q^\pi(s_t, a)$, the empirical ordering approaches the corresponding ordering of Q^π as the number of branch continuations increases.

Our implementation uses the same ordering signal, but does not force every pair into a binary preference. Instead, we train the model with three possible outputs:

$$\begin{cases} \texttt{A}, & \widehat{Q}_M(s_t, a_A) - \widehat{Q}_M(s_t, a_B) > \gamma_e, \\ \texttt{B}, & \widehat{Q}_M(s_t, a_B) - \widehat{Q}_M(s_t, a_A) > \gamma_e, \\ \texttt{T}, & \left|\widehat{Q}_M(s_t, a_A) - \widehat{Q}_M(s_t, a_B)\right| \le \gamma_e. \end{cases}$$

Thus, `A` and `B` represent the two directions of the same pairwise ordering used in Eq. 8, while `T` represents an empirical tie region in the supervision; at deployment, such a prediction contributes no decisive winner. When $\gamma_e = 0$, this supervision directly reflects the empirical strict ordering: unequal branch estimates produce a preference for the action with the larger estimated continuation value, while equal estimates are labeled `T`. For ALFWorld and τ^3-Retail, branch outcomes are binary, so we use $\gamma_e = 0$: outcomes 1 versus 0 produce a decisive preference, whereas 1 versus 1 and 0 versus 0 produce a tie. For WebShop, whose returns provide graded numerical feedback, we use a positive margin $\gamma_e = \gamma$. This deliberately treats small empirical return differences as ties rather than forcing the model to learn a preference from a weak distinction. The margin is therefore a conservative implementation-level surrogate for the strict target in Eq. 4: sufficiently separated action pairs are trained according to their empirical ordering, whereas pairs within the margin contribute no decisive preference.

This construction connects directly to the target in the main method. Because $\widehat{Q}_M(s_t, a) \to Q^\pi(s_t, a)$ as the number of independent branch continuations increases, the training signal increasingly reflects the ordering of the true actor-conditioned continuation values. With $\gamma_e = 0$, it approaches the strict relation in Eq. 4 away from exact ties. With $\gamma_e > 0$, the implementation intentionally abstains on sufficiently small value differences rather than attempting to resolve them.

Binary comparator at inference. Although the model is trained with the three outputs $\{\texttt{A}, \texttt{B}, \texttt{T}\}$, the runtime gate in Section 3.2 requires only the binary judgment in Eq. 4. We obtain this judgment by evaluating each action pair in both input orders.

Specifically, we set:

$$C_\theta(s_t, a_A, a_B) = \begin{cases} 1, & \text{the model predicts } \texttt{A} \text{ for } (s_t, a_A, a_B), \\ & \text{and predicts } \texttt{B} \text{ for } (s_t, a_B, a_A), \\ 0, & \text{otherwise.} \end{cases} \tag{39}$$

Thus, a_A is counted as defeating a_B only when both input orders express the same semantic preference for a_A. A tie prediction, or a pair of predictions that is inconsistent under input reversal, produces $C_\theta = 0$. Eq. 39 is the concrete implementation of the binary C_θ used in the main method. In particular, the runtime prediction in Eq. 5 is exactly $\widetilde{B}_i = C_\theta(s_t, A_i, a_t)$, after applying the bidirectional check above. The winner count in Eq. 6 and the candidate-relative estimate in Eq. 10 therefore operate on

the same binary comparator defined in Section 3.2; the ternary outputs are only an implementation detail used to obtain that decision conservatively.

We study the empirical effect of ternary supervision and bidirectional inference in Appendix C.1.

B.5 Selection of K and R

K controls the number of alternatives reviewed at a state and R controls how many stable winners are required to reject the proposal. We select K and R using end-to-end performance on held-out validation tasks and freeze the selected values before test evaluation. We use the same K and R across actor models within each environment.

Table 6: Validation performance under different settings of K and R on WebShop.

Configuration	K	R	Mean reward	Success (%)
COTA, cautious	4	4	0.441	18.2
COTA, medium	8	2	0.555	28.4
COTA, active	4	1	**0.635**	**34.2**

After our grid search on respective held-out validation tasks, we select K=4, R=1 for WebShop and ALFWorld, and K=4, R=2 for τ^3-Retail. These values are then fixed for all test actors.

B.6 Reference Candidate Mechanisms

Environment actions. When the environment exposes legal actions, we sample without replacement. In WebShop these are current-page clicks and search; in ALFWorld they are admissible text commands.

Offline actions. Offline retrieval draws concrete actions observed in branch data under a compatible decision condition. Retrieved actions are rebound to currently observed entities and rejected if required identifiers or policy prerequisites are absent. Retail uses this source to recover useful tool patterns that are unlikely to appear in a small online sample.

Small-LM actions. A small LLM (Qwen2.5-0.5B-Instruct) receives the actor-visible state and proposes alternatives under several decision modes, such as information gathering, progress, recovery, and a distinct alternative. The generator never votes on its own actions.

Table 7: Default candidate source by benchmark and actor family.

Environment	Actor condition	Candidate source
WebShop	all actors	environment
ALFWorld	Qwen3-8B	environment
ALFWorld	stronger actors	small LM + environment
τ^3-Retail	all actors	small LM + offline

B.7 Baseline Implementations

Self-Reflection. The actor reviews each proposed action before execution, including both user-facing responses and tool calls in Retail. It performs one conservative self-check and may retain or revise the proposal. Retaining an action also preserves its original Thought or response content. Only the final executed action and its resulting observation enter the persistent interaction history.

Asym-AC. We implement Asym-AC with a fixed, zero-shot Qwen3-8B critic. The critic is invoked at environment-specific intervention points: purchase actions in WebShop; object manipulation and state-changing actions in ALFWorld; and database-mutating tool calls in τ^3-Retail. At each reviewed step, the critic provides one round of free-form feedback, after which the actor replans once. The revised action is not reviewed recursively. These WebShop and ALFWorld intervention points are our benchmark-specific adaptations rather than settings prescribed by the original work.

AgentRM. We implement the AgentRM inference-time search baseline using beam search with $W_1 = W_2 = K$ and $K = 4$. At each expansion, retained prefixes are extended with up to K candidate actions, scored by the released AgentRM reward model, and pruned to the highest-scoring K prefixes. The final trajectory is selected according to the AgentRM scores.

Scope of comparison. For Jiang et al. (2026), we use the zero-shot Asym-AC variant because no verified author-trained SFT critic checkpoint was available to us. We do not directly reproduce the action-conditioned controller of Zhang et al. (2026), as we could not identify a public implementation of the controller required to reproduce their intervention setting.

The episode-level budget in Section 4.6 is used only as a diagnostic for COTA and is not imposed on the baselines in Table 1. Self-Reflection reviews every proposed action before execution. For Asym-AC, we found that per-step criticism performs poorly, so we follow its original design more closely and invoke the critic only at selected consequential decision points.

B.8 Cost Accounting

B.8.1 Offline computation

Offline cost includes base rollout generation, branch continuations, and comparator fine-tuning and is reported in Table 8. It is amortized across every actor and every future evaluation that reuses the comparator, and is therefore kept separate from the online Avg. T values in Table 1.

Table 8: Offline cost accounting. The time unit is H200 GPU-hour.

Environment	Base rollout	Branch rollout	Comparator FT	Total GPU-hours
WebShop	1.48	78.65	1.23	81.36
ALFWorld	1.97	28.39	0.80	31.15
τ^3-Retail	0.27	7.79	0.36	8.42

B.8.2 Online API-equivalent cost

We report API-equivalent inference cost based on the input and output tokens recorded during evaluation. For each model m, the cost is computed as

$$C_m = \frac{N_{\text{in}}^{(m)} p_{\text{in}}^{(m)} + N_{\text{out}}^{(m)} p_{\text{out}}^{(m)}}{10^6},$$

where $N_{\text{in}}^{(m)}$ and $N_{\text{out}}^{(m)}$ denote the total input and output token counts, and $p_{\text{in}}^{(m)}$ and $p_{\text{out}}^{(m)}$ are the corresponding OpenRouter list prices per million tokens. We do not apply caching discounts or provider-specific promotions. Table 9 lists the prices used in our accounting.

Table 9: API prices used for inference-cost accounting. Prices are in USD per million tokens and correspond to the OpenRouter list prices.

Model	Input ($/M)	Output ($/M)
Qwen3-8B	0.117	0.455
Qwen3.6-35B-A3B	0.050	0.700
DeepSeek-V4-Flash	0.068	0.168
Llama-3.1-8B-Instruct (AgentRM)	0.020	0.040

AgentRM Xia et al. (2025) uses fine-tuned Llama-3.1-8B-Instruct as the backbone of its reward model with a scalar prediction head. Since reward-model evaluation scores an input trajectory without autoregressive token generation, we approximate its API-equivalent cost by charging all tokens processed by the reward model at the backbone's input-token rate, with no output-token charge.

COTA additionally invokes a 0.5B comparator. We run this comparator locally and do not assign it an external API charge. OpenRouter does not list Qwen2.5-0.5B-Instruct, while a closely matched

model at the same scale, Qwen3-0.6B, is available through a free endpoint. We therefore exclude the Qwen2.5-0.5B-Instruct from the reported API-equivalent monetary cost. This convention concerns external API pricing only and does not imply zero underlying hardware cost.

C Additional Experimental Results

C.1 A/B versus A/B/T

Table 10: Comparator-target ablation on 100 held-out WebShop tasks. Both rows use a bidirectional gate and environment-only $K{=}4, R{=}1$ candidates.

Training target	Pref. consistency	Valid output	End reward	Zero reward
A/B only	39.51%	86.16%	0.5031	37.0%
A/B/T	**57.70%**	**99.53%**	**0.5442**	**22.0%**

Explicit ties improve order consistency and make abstention available when the branch outcomes do not justify a winner.

C.2 Absolute-Q Diagnostics

Table 11: Validation quality of the three Qwen2.5-0.5B absolute-Q critics.

Environment	Pairwise ranking acc. (%)	Spearman	BCE
WebShop	69.13	0.618	0.527
ALFWorld	64.23	0.366	0.635
τ^3-Retail	68.20	0.423	0.639

Table 12: Online intervention frequency (%) for the absolute-Q critic. Forced reports states where argmax replaces the actor proposal; selective reports gate rounds that request actor replanning.

	Qwen3-8B		Qwen3.6	
Environment	Forced	Selective	Forced	Selective
WebShop	61.1	46.5	48.8	26.6
ALFWorld	83.6	50.5	87.0	56.5
τ^3-Retail	77.7	38.7	66.0	32.9

As shown in 11, the Absolute-Q scalar critic can show reasonable held-out ranking accuracy while remaining poorly calibrated for online control. As shown in 12, in ALFWorld it frequently scores several legal but subgoal-irrelevant moves above the proposal a_t, causing forced selection to replace more than 80% of actor actions. Constructive intervention absorbs some of these mistakes, but direct pairwise supervision remains substantially more effective in the closed loop.

C.3 Cross-Actor Preference Changes

We test how pairwise action preferences change when the branch-point state and actions are fixed but the continuation actor is replaced. For each environment, we sample 24 held-out prefixes from Qwen3-8B trajectories and replay the same branch-point actions with different continuation actors. This yields 240 sibling action pairs for WebShop, 240 for ALFWorld, and 211 for τ^3-Retail. We first evaluate each pair with Qwen3-8B as the continuation actor and keep only non-tied pairs, leaving 106 pairs on WebShop, 48 on ALFWorld, and 70 on Retail. We then evaluate the same pairs with Qwen3.6-35B-A3B or DeepSeek-V4-Flash. Table 13 reports whether the Qwen3 preference is preserved (*Same*), becomes indistinguishable (*Tie*), or changes direction (*Reverse*). Percentages are computed over the non-tied Qwen3 pairs in each environment.

Environment	Actor shift	Same	Tie	Reverse
WebShop	Qwen3-8B → Qwen3.6-35B-A3B	17.0%	79.2%	3.8%
ALFWorld	Qwen3-8B → Qwen3.6-35B-A3B	39.6%	60.4%	0.0%
ALFWorld	Qwen3-8B → DeepSeek-V4-Flash	6.3%	93.8%	0.0%
τ^3-Retail	Qwen3-8B → Qwen3.6-35B-A3B	42.9%	55.7%	1.4%
τ^3-Retail	Qwen3-8B → DeepSeek-V4-Flash	44.3%	52.9%	2.9%

Table 13: Changes in pairwise preferences after replacing the Qwen3 continuation actor. Each row is computed over action pairs that are non-tied under Qwen3.

Table 14: Effect of the reference candidate mechanism on WebShop. Mean reward and exact success rate are reported on the same 500 held-out tasks.

	Qwen3-8B		Qwen3.6-35B-A3B	
Reference candidate mechanism	Mean reward	Success rate	Mean reward	Success rate
Environment-only	0.563	27.8 %	0.681	56.6 %
Actor sampling + environment	0.602	30.4 %	0.694	57.4 %
Counterfactual prompting + environment	0.579	27.0 %	0.674	56.0 %
Small-LM sampling + environment	0.609	28.8 %	0.696	57.2 %
SFT small-LM sampling + environment	0.583	28.8 %	0.673	54.6 %

Most changes are ties rather than reversals. Across the five actor shifts in Table 13, 234 of 342 pair–actor comparisons (68.4%) become ties, while only 7 (2.0%) reverse direction. Thus, pairwise preferences are actor-dependent, but replacing the continuation actor more often removes a preference than reverses it.

One interpretation is that a stronger continuation actor can recover from either branch-point action and reach the same outcome, even when the distinction matters for Qwen3-8B. The high tie rates are consistent with this behavior. Accordingly, the cross-actor gains in Table 1 should be interpreted as reuse of the COTA pipeline across actors, rather than as evidence that the underlying Q^π ordering is invariant to the continuation policy. Each branch is evaluated with a single continuation rollout, so this analysis concerns empirical outcome orderings rather than exact Q^π values.

C.4 Effect of Candidate Support: A WebShop Study

COTA can only compare alternatives exposed by the reference candidate mechanism. We therefore isolate the effect of candidate support on WebShop by varying only this mechanism, while keeping the comparator, intervention rule, candidate budget ($K = 4$), and 500 held-out tasks fixed. Environment and small-LM candidates follow the construction in Section 4.2 and Appendix B.6. For the additional variants, actor sampling draws high-temperature alternatives from the deployment actor, counterfactual prompting asks the same actor to generate a different next action, and the SFT small-LM uses the same 0.5B backbone fine-tuned on training-set prefix-branch data.

Table 14 shows that the reference candidate mechanism has a measurable effect on COTA's performance even when the comparator and intervention rule are fixed. Generic small-LM generation achieves the highest mean reward for both actors, while direct actor sampling attains the highest exact success rate. Other mechanisms provide smaller or inconsistent gains: counterfactual prompting offers little improvement over environment-only candidates, and fine-tuning the 0.5B proposer does not further improve performance in this setting.

D Prompt Templates

D.1 Actor Interaction Templates

We summarize the actor prompts used in each environment below. Braced fields denote runtime content, and benchmark-provided policies or fixed few-shot demonstrations are inserted verbatim

where indicated. We show the common prompt structure for each benchmark and describe actor-specific deviations separately.

WebShop. The Qwen actors receive a fixed shopping instruction with few-shot demonstrations, followed by the executed history and current page state:

```
[System]
You are a shopping agent operating in a text-only WebShop
environment.  Follow the user instruction and purchase the
correct product.
Valid actions are search[<keywords>] and click[<exact
text>].
Reply with exactly:
Thought:  <brief reasoning>
Action:  <one executable action>
{fixed few-shot demonstrations}
[User]
Task:  {shopping instruction}
Execution history:  {executed action--observation history}
Current observation:  {current page}
Available actions:  {current executable actions}
```

Actor-specific variant. Qwen3-8B and Qwen3.6-35B-A3B use the rendered execution summary above. For DeepSeek-V4-Flash, we preserve the archived WebShop prompt and provide the interaction as the raw ReAct message history instead.

ALFWorld. Actors receive the household task, two fixed task-family demonstrations, and the executed interaction history:

```
[System]
You are a household agent operating in the text-only
ALFWorld environment.  Complete the task using one currently
admissible command per turn.
Reply with exactly:
Thought:  <brief grounded reasoning>
Action:  <one admissible command>
Do not invent entities or issue commands outside the
admissible action set.
{two task-family few-shot demonstrations}
[User]
Task:  {household task}
Initial observation:  {initial observation}
{executed Thought/Action--Observation history}
Current observation:  {current observation}
```

τ^3-Retail. Retail follows the benchmark-native customer-service interface. The system message contains the domain policy, while customer messages, assistant responses, and tool outputs retain their original API roles:

```
[System]
You are a customer-service agent.  Follow the supplied
policy exactly.  On each turn, either send one
natural-language response or make one tool call.  Do not
fabricate information, and obtain explicit confirmation
before consequential database operations.
<policy>
{benchmark domain policy}
</policy>
[Conversation]
{original user/assistant/tool interaction history}
```

Tool names, descriptions, and argument schemas are provided through the native API tool interface rather than inserted into the text prompt. Unlike WebShop and ALFWorld, Retail does not require a textual `Thought:`/`Action:` response format.

D.2 Comparator Prompt Template

The comparator uses a prompt template across all three environments. Only the rendered state representation is benchmark-specific. Table 15 summarizes the information exposed to the comparator; in all cases, only executed interaction history is included, and unexecuted actor reasoning is excluded.

Table 15: Benchmark-specific state information provided to the comparator.

Environment	State representation
WebShop	Task, executed history, current page, executable search/click actions
ALFWorld	Task, executed action–observation history, current observation, admissible actions
τ^3-Retail	Policy and interaction state, including grounded entities, confirmations, tool state, and recent dialogue

The resulting state block is inserted into the following shared prompt:

```
[System]
You are an expert judge for an interactive agent.  Given the
task, observable progress, current state, and two candidate
next actions, decide which action is more likely to lead to
correct task completion when the same base agent continues.
If one is clearly better, answer A or B. Otherwise, answer
T.

[User]
# Task and observable trajectory
{benchmark-specific state}
# Candidate actions
A: {action A}
B: {action B}
# Answer
Reply with exactly one letter:  A, B, or T.
```

D.3 Small-LM Candidate Generation

This prompt is used only when the reference candidate mechanism includes small-LM generation; environment-sampled and offline-retrieved candidates do not use a generation prompt. The generator produces executable alternatives but does not score them or decide whether intervention should occur. We use benchmark-specific generation perspectives to encourage diverse alternatives: product discovery, requirement verification, configuration, and progress in WebShop; subgoal progress, information gathering, recovery, and alternative actions in ALFWorld; and clarification, verification, confirmation, execution, recovery, alternatives, and completion in τ^3-Retail.

```
[Context]
{task, observable interaction history, current state, and
available actions or tools}

[User]
Current actor proposal:  {proposed action}
Generation perspective:  {benchmark-specific instruction}

Generate one executable alternative next action from this
perspective.  Do not judge whether the current proposal is
correct or whether intervention should occur.  Use only
information available in the current state, and do not
invent entities, tool arguments, or observations.  Return
an action different from the current proposal; if no useful
alternative is available, output NONE.

{benchmark-specific output format}
```

For WebShop and ALFWorld, the generator returns one executable textual action. For τ^3-Retail, it returns either one user-facing response or one native tool call with grounded arguments. Exact benchmark-specific formatting is provided with the corresponding environment implementation.

D.4 Prompt Templates of Baselines

D.4.1 Self-Reflection Prompt Template

The actor receives one conservative self-check before execution. It may keep its original proposal.

```
[User: Self-check]
Unexecuted proposal: {original Thought and Action, or
response/tool call}

Review this draft against the task, observed interaction
history, current state, and applicable constraints. Default
to keeping it unless there is a concrete error. Do not
invent a mistake merely to offer a different plan. An
information-gathering action need not already satisfy every
final task requirement.

Return the action that should actually be executed. If
keeping the original action, preserve its original Thought
exactly. Otherwise, give a grounded reason and one
executable action in the actor's normal format.
```

For Retail, keeping a proposal preserves its response text or tool name and arguments. The prompt explicitly forbids treating a proposed tool call as completed.

D.4.2 Asym-AC Critic Prompt Template

Critic prompt. The critic reviews the proposed action against observable evidence. Retail additionally supplies the domain policy and concrete tool arguments.

```
[System]
You are an action-verification critic. Ground your review
in the supplied task, executed interactions, current
observation, available actions, and applicable policy.
Identify concrete errors or confirm that the proposal is
appropriate.

The proposal has NOT been executed. Missing evidence is
not evidence of a violated constraint. Do not invent
commands, requirements, or unobserved state transitions.
Give concise, actionable feedback.

[User]
Task and applicable policy: {context}
Executed interaction history: {prefix}
Current observation and available actions: {state}
Unexecuted actor proposal: {proposal}
```

Actor revision prompt. The actor receives the critic's feedback as:

```
[User: Critic feedback]
Unexecuted draft: {proposal}
Feedback: {critic feedback}

The draft has NOT been executed. The environment and latest
observation are unchanged. Critic feedback is advice, not
an observation or proof of success. Revise the draft in
light of this feedback; do not skip ahead to the action that
would follow it.

Return the action that should actually be executed in the
current state. You may repeat the original proposal if the
```

```
feedback confirms it, preserving its original Thought.  Use
the actor's normal output format.
```

For Retail, the revision requests one response or native tool call and explicitly states that no tool result exists for the unexecuted draft.

D.4.3 AGENTRM SCORING INPUT

AgentRM receives a serialized interaction prefix rather than a free-form critique request. Its input layout is summarized below; the scalar score is read from the reward head, not generated as text.

```
[User]
{environment instructions; policy and tool descriptions for
Retail}
[Assistant]
OK
[User]
{initial observation or customer request}
[Assistant]
{actor response or executed action}
[User]
{resulting environment observation or customer response}
...
[Assistant]
{latest response or action in the candidate prefix}
[User]
{resulting observation or response}
```

D.4.4 ABSOLUTE-Q ABLATION PROMPT TEMPLATE

The absolute-Q ablation uses our COTA's same observable information as the pairwise comparator, but scores one state–action pair at a time.

```
[System]
You are a process reward model for an interactive agent.
Given the observable task state and a candidate next action,
estimate how likely the action is to lead to successful task
completion when the same actor continues.

[User]
# Task and observable state
{benchmark-specific state}
# Candidate action
{candidate action}
```

The benchmark-specific state renderer follows the same observable-state construction as the comparator. The model outputs a scalar score through its classification head, with no textual rationale.

D.4.5 COTA FEEDBACK PROMPT TEMPLATE

The comparator never writes free-form criticism. If at least R candidates win, the actor receives a short, structured message identifying its unexecuted proposal and listing preferred alternatives in ranked order. The actor is explicitly allowed to keep the original action or synthesize a third action.

```
[ADVISOR FEEDBACK]
The proposed action {proposal} may not be the best next step.
Preferred alternatives:  1.  {winner 1}; 2.  {winner 2}.
Reconsider the current state and output the action that
should actually be executed.  The recommendations are
non-binding.
```