ARCO: ADAPTIVE RUBRICS WITH CO-EVOLUTION FOR MULTI-STEP LLM-BASED AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning for multi-step LLM agents often relies on scalar rewards that indicate success but cannot explain why a trajectory is good or bad. Rubric-based rewards improve interpretability through natural-language criteria, but many existing approaches score *complete trajectories* with a frozen closed-source judge, limiting step-level guidance and adaptation during training. We propose **ARCO** (**A**daptive **R**ubric **CO**-evolution), which generates interpretable, action-specific per-step rubrics and dense rubric-conditioned rewards, then continuously co-evolves the rubric model with the agent on shared on-policy rollouts. After warmup, trajectory decomposition fits step rewards to terminal outcomes without step labels or an external judge during online training. Across HotpotQA, 2WikiMultiHopQA, and MuSiQue with two open-source backbones, ARCO achieves the highest EM in all settings over outcome-, rubric-, and process-reward baselines. We further analyze ARCO along several dimensions—the action-specificity of its rubrics, the number of criteria per step, and the rubric model's scale and family—showing that it yields an interpretable step-level signal useful for diagnosing agent behavior. Code and data are available at `https://anonymous.4open.science/r/arco-anon-2C61`.

1 INTRODUCTION

Reinforcement learning (RL) has become a standard approach for improving multi-step LLM-based agents on tasks such as multi-hop QA and tool use (Zeng et al., 2025; Kalyan & Andrews, 2025; Jin et al., 2025). Most methods train with black-box scalar rewards—either trajectory-level outcome rewards or step-level process rewards—that indicate quality but cannot explain *why* a trajectory is good or bad (Ouyang et al., 2022; Lightman et al., 2024; Xi et al., 2026). This coarse and opaque supervision makes credit assignment across steps difficult, especially over long decision horizons, and can leave the learning process vulnerable to reward hacking (Gao et al., 2023).

Rubric-based rewards ground evaluation in explicit natural-language criteria (Gunjal et al., 2026; Gupta et al., 2025; Shao et al., 2025; Liu et al., 2026a). Yet existing evaluators lack key properties for multi-step credit assignment. Most score at trajectory level, leaving per-step credit untouched; and whether the rubric is static (Gunjal et al., 2026), generated per trajectory (Gupta et al., 2025), or evolved via an on-policy buffer (Shao et al., 2025), judging relies on a frozen closed-source LLM (e.g., GPT-4). Step-level interpretable PRMs exist (Chae et al., 2026; Zhang et al., 2026b), but they too fix the evaluator before policy learning. To our knowledge, prior work does not combine per-action rubric generation with a trainable step-level evaluator that co-evolves with the policy.

Consider a multi-hop question: *In what year was the university where Sergei Tokarev was a professor founded?* A trajectory-level outcome reward assigns the same terminal reward to a trajectory that simply guesses "1755" and one that first identifies Tokarev's university before looking up its founding year. A static rubric might check "did the agent search?"—but once the agent learns to search, its errors shift to subtler modes: targeting the wrong entity, redundant retrieval, or premature answers. Updating the rubric text can reflect these evolving failure modes, but a frozen judge cannot adapt its scoring parameters to the policy's continually evolving behavior during training.

We argue that an effective reward model for multi-step agents should be (i) *rubric-based*, providing interpretable criteria rather than opaque scalars; (ii) *step-level*, assigning per-action scores rather than a single trajectory score; and (iii) *co-evolutionary*, adapting the rubric model alongside the policy at the parameter level. We propose **ARCO** (**A**daptive **R**ubric **CO**-evolution), a single trainable

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

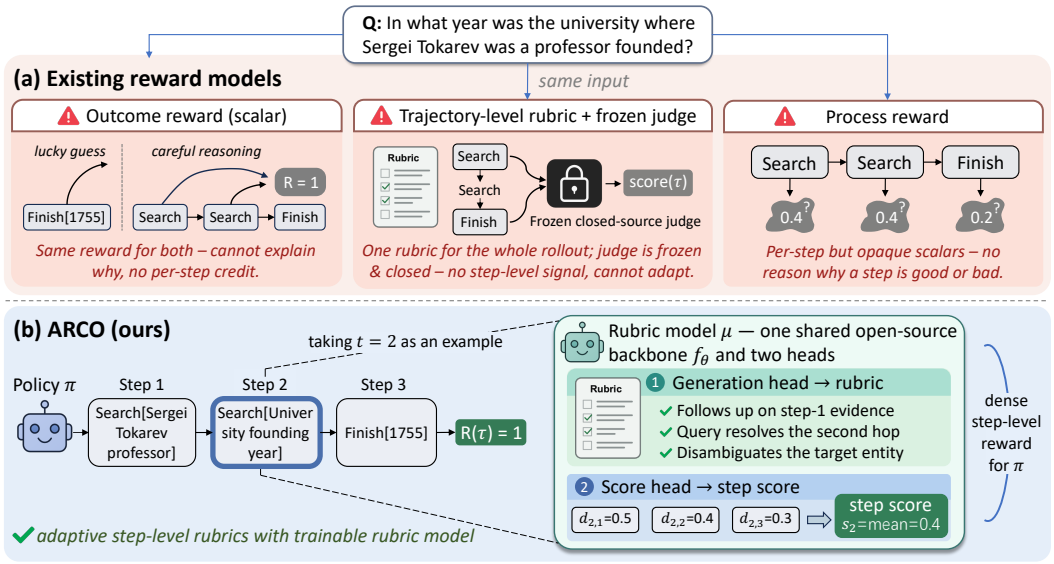


Figure 1: Reward signals for a multi-hop QA agent. **(a)** Existing reward models each fall short: *outcome reward* is a single scalar with no per-step credit; a *trajectory-level rubric* scored by a frozen judge cannot adapt to the policy; and a *PRM* gives per-step but opaque scores. **(b)** ARCO uses a single open-source rubric model μ to generate a natural-language rubric per action and score the action against it, yielding an interpretable, step-level, adaptive reward with no external judge.

rubric model μ that couples two complementary functions on a shared backbone: a *generation head* producing per-step natural-language criteria and a *score head* predicting rubric-conditioned step rewards. Both μ and π are updated jointly on on-policy data, allowing rubrics and scores to adapt with the policy. Figure 1 contrasts ARCO with existing outcome, trajectory-rubric, and process rewards.

Our main contributions are:

- We introduce **ARCO**, an adaptive, *step-level* rubric-based process reward framework in which both the rubric generator and the scoring function co-evolve with the policy at the parameter level throughout training, removing the dependence on a frozen closed-source judge.
- We propose a unified rubric model μ that couples rubric generation and action scoring into a single, jointly optimizable objective on a shared backbone so that the rubric and the reward are learned together throughout training, rather than as two separately optimized, decoupled stages.
- We conduct extensive experiments demonstrating ARCO's effectiveness, along with comprehensive analyses that inform future work on interpretable process rewards.

2 Preliminaries

A multi-step LLM-based agent solves a task $q \in \mathcal{Q}$ by sequentially selecting actions $a_t \sim \pi(\cdot \mid h_t)$ given a trajectory prefix $h_t = (q, a_1, o_1, \ldots, a_{t-1}, o_{t-1})$, producing a trajectory τ that receives a binary outcome reward $R(\tau) \in \{0, 1\}$ only at termination. The dominant paradigm optimizes π directly against this terminal signal with an *outcome reward model* (ORM) (Jin et al., 2025; Song et al., 2025); but because the reward is sparse and trajectory-level, it says nothing about *which* intermediate action helped or hurt, making credit assignment hard. To densify the signal, a *process reward model* (PRM) assigns a scalar score s_t to each step (Xi et al., 2026); yet standard PRMs compress multiple aspects of step quality into a single opaque scalar, revealing neither which criterion an action violated nor why. Rubric-based rewards make this explicit by pairing the score with natural-language criteria. Most existing rubric-based methods apply them at the trajectory level; the per-step form we adopt couples a rubric generator ρ with a rubric-conditioned scorer g,

$$r_t = \rho(h_t, a_t), \qquad s_t = g(h_t, a_t, r_t). \tag{1}$$

Existing methods differ along three axes. *Evaluator adaptation*: criteria may be fixed before training (Gunjal et al., 2026; Liu et al., 2026a), generated per trajectory by a frozen generator (Gupta et al., 2025), or evolved in an on-policy buffer (Shao et al., 2025)—but the scoring parameters

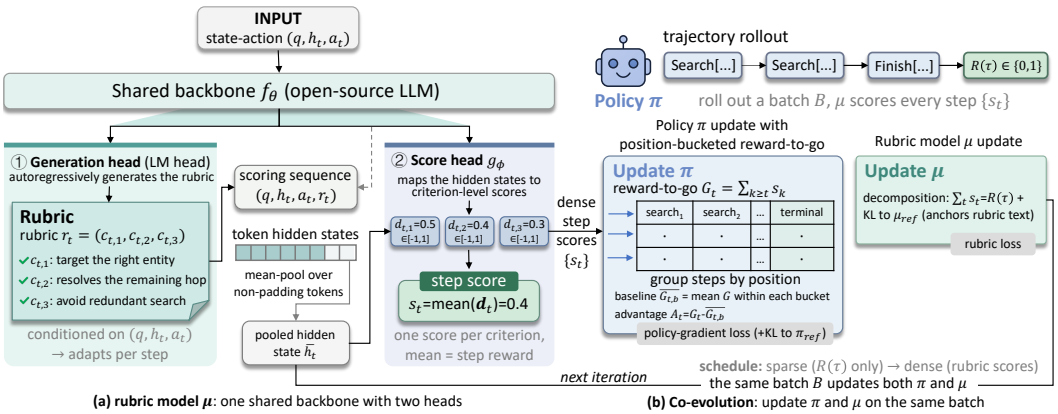


Figure 2: Overview of ARCO. **(a) Rubric model** μ: from the state–action input at step t, a single open-source backbone with two heads produces a per-step rubric and score—a *generation head* writes the rubric r_t as a natural-language checklist, and a *score head* scores each criterion and averages them into the step score s_t. **(b) Co-evolution**: on the *same* batch, π is updated using the reward-to-go derived from $\{s_t\}$ with a position-bucketed baseline (Eq. 3), and μ by a trajectory-decomposition loss with a KL that anchors the rubric text (Eq. 4); both models thus co-evolve on shared rollouts.

remain fixed, and criteria are typically shared across steps. *Scoring granularity*: g usually returns a trajectory-level score rather than per-step s_t; step-level interpretable scorers exist in domains such as web navigation (Chae et al., 2026; Zhang et al., 2026b), though there too the evaluator is fixed before policy training. *Judge model*: g is usually a closed-source LLM and therefore cannot co-evolve with π. ARCO jointly addresses all three axes with a trainable rubric model μ whose two branches instantiate ρ and g on a shared backbone: ρ produces context-specific criteria *per step*, g predicts *step-level* scores conditioned on them, and μ co-evolves with π on on-policy data throughout training.

3 ARCO

We study an LLM-based agent that solves a multi-hop question $q \in \mathcal{Q}$ over multiple steps, following the notation of Section 2: at step t the policy π selects an action a_t from the prefix h_t, and the trajectory τ receives a binary outcome reward $R(\tau) \in \{0, 1\}$ only at the end. ARCO turns this sparse signal into interpretable process supervision by attaching a natural-language rubric r_t and a scalar score s_t to the action at each step. Unlike pipelines that read rubrics with a frozen external judge, ARCO instantiates the generator ρ and the scorer g as two heads of a single trainable model μ that co-evolves with π. Figure 2 illustrates our method and its co-evolution training procedure.

3.1 Rubric Model

The rubric model μ follows two design principles. First, evaluation criteria should be *generated* rather than fixed, because step quality is operation-dependent and its failure modes shift as the policy evolves: a `Search` may fail by wrong-entity targeting or redundancy, a `Finish` by missing support or wrong format, and a static checklist goes stale. We let a language model write a fresh, context-specific rubric per step, making the evaluator adaptive. Second, the rubric and score should improve together rather than come from disconnected modules, so we place generation and scoring in a *single* model on a shared backbone, where the scoring gradient reshapes the representation that rubric generation also draws on. Concretely, μ adds two heads to a causal LM backbone f_θ: a *generation head* that produces the rubric r_t, and a *score head* g_ϕ that rates how well the action satisfies each criterion and outputs the step score s_t. In effect, μ writes a local checklist for the current prefix–action pair and scores against it, making the dimensions of evaluation explicit as readable criteria.

We now specify each head in turn. The generation head acts as the rubric generator: given (h_t, a_t), it generates an ordered rubric $r_t = (c_{t,1}, \ldots, c_{t,K}) \sim p_\theta(\cdot \mid h_t, a_t)$, an ordered list of K criteria that each name one aspect of action quality. Because the rubric is generated after observing the prefix, it adapts across stages: early searches are judged by entity targeting and evidence coverage, later searches by whether they resolve the remaining hop, and finish actions by answer support and format.

The score head acts as the evaluator: it reads the full scoring sequence (h_t, a_t, r_t) through the shared backbone, forms $\bar{\mathbf{h}}_t$ by attention-mask mean pooling over the non-padding hidden states, and maps it to K criterion-level scores whose mean is the step reward,

$$\mathbf{d}_t = g_\phi(\bar{\mathbf{h}}_t) \in [-1, 1]^K, \qquad s_t = \frac{1}{K} \sum_{j=1}^{K} d_{t,j}, \tag{2}$$

where $d_{t,j} \in [-1, 1]$ rates how well action a_t satisfies criterion $c_{t,j}$, $\mathbf{d}_t$ collects these K scores, and their mean s_t is the step-level reward for the policy (Figure 2a). Sharing the backbone lets both heads improve together: since s_t is predicted from $\bar{\mathbf{h}}_t = f_\theta(h_t, a_t, r_t)$, the scoring gradient updates both g_ϕ and the backbone θ producing the rubric, so pressure to score accurately reshapes the representation that generation depends on. This coupling is confined to *optimization*: because r_t is a discrete sample, the gradient does not directly reach the rubric *text*, which is instead anchored by warmup imitation and a KL term (§3.2). Splitting μ into separate generator and scorer would remove this shared-representation coupling and require an explicit mechanism to coordinate their adaptation.

3.2 Co-Evolutionary Training

We warm up both models by supervised fine-tuning (SFT) on rollouts from a closed-source LLM teacher in the same retrieval environment: π imitates the actions of high-reward trajectories, while μ learns to generate the annotated criteria and regress their scores, projected so that the per-step means satisfy the trajectory decomposition $\sum_t s_t = R(\tau)$ (Appendix A). The resulting checkpoints initialize the evolving π and μ; frozen copies serve as the references π_{ref} and μ_{ref} used by the KL terms below, and the same warmup is applied to all baselines for fairness. The remainder of this section describes the on-policy co-evolution loop (Figure 2b), which updates π and μ on the *same* rollouts so that the evaluator tracks the policy's shifting failure modes.

Each iteration proceeds in three steps: the current π rolls out a batch $\mathcal{B}$ of trajectories, the current μ annotates every active step with a rubric and a score, and both models are updated on this shared batch. Training follows a *sparse-to-dense* schedule: the first few epochs optimize π against the outcome reward $R(\tau)$ alone, giving μ enough time to adapt to the fresh on-policy rollouts; thereafter, policy updates use the dense rubric scores for the remaining training. The policy is updated by credit assignment on the dense scores. Within a trajectory, we use the reward-to-go $G_t = \sum_{k \geq t} s_k$, which propagates the consequences of an action forward to the steps it enables. A global mean baseline is ill-suited here, since reward-to-go varies systematically by position: earlier steps aggregate more future scores than later ones, even before accounting for differences in action quality. A single average would therefore confound position with quality. We instead center each reward-to-go on a *position-bucketed* baseline $\bar{G}_b$, the mean reward-to-go over all steps in the batch that share bucket b, where a step's bucket is its search rank or `terminal` for the trajectory's last action (a successful `Finish` or a forced finish at budget exhaustion). The resulting advantage $A_t = \alpha\,(G_t - \bar{G}_{b(t)})$, with α scaling the dense reward, enters a KL-regularized policy-gradient objective,

$$\mathcal{L}_\pi^{\text{rl}} = -\frac{1}{|\mathcal{B}|} \sum_{\tau \in \mathcal{B}} \sum_{t \in \mathcal{V}_\tau} A_t \, \log p_\pi(a_t \mid h_t) + \lambda_\pi \, \mathcal{L}_{\text{KL}}^{\pi}, \tag{3}$$

where $\mathcal{V}_\tau$ is the set of valid policy steps in τ, and λ_π weights a token-level KL term $\mathcal{L}_{\text{KL}}^{\pi}$ against the frozen reference π_{ref}, limiting drift from the warmup policy. On the same batch, μ is updated by a trajectory-decomposition loss with a stabilizing rubric-anchoring KL term,

$$\mathcal{L}_\mu^{\text{rl}} = \frac{1}{|\mathcal{B}|} \sum_{\tau \in \mathcal{B}} \left[w_\tau \Big(\textstyle\sum_t s_t - R(\tau) \Big)^2 + \lambda_\mu \mathcal{K}_\tau \right]. \tag{4}$$

Here w_τ balances successful and unsuccessful trajectories, and $\mathcal{K}_\tau$ is the rubric-token KL between the current model and its frozen reference, averaged over rubric tokens per step and summed over steps (Appendix A.3). The decomposition term fits the summed step scores to the observed outcome without gold step labels during RL. The constraint alone does not uniquely identify individual step rewards. Warmup supervision and rubric-conditioned scoring provide the main inductive bias toward meaningful decompositions, while the KL keeps generated rubrics close to the warmup language; without it, shared-backbone updates driven by the decomposition objective could cause the rubric language to drift. As the policy changes, its rollouts expose a shifting distribution of failure modes to the evaluator, whose updated scores reshape the rewards driving subsequent policy updates, so the two co-evolve throughout training. Algorithm 1 in Appendix A.4 gives the full loop.

4 EXPERIMENTS

We evaluate ARCO on three multi-hop QA benchmarks across two open-source backbone families. We organize the analysis around five research questions:

- **RQ1.** Does ARCO outperform existing outcome-, rubric-, and process-reward baselines?
- **RQ2.** How does each of ARCO's design choices contribute to performance?
- **RQ3.** How specifically are ARCO's rubrics tied to the actions they evaluate?
- **RQ4.** How does rubric width K affect rubric quality and downstream performance?
- **RQ5.** How do the architecture and scale of the rubric-model affect policy learning?

4.1 EXPERIMENTAL SETUP

Datasets. We evaluate on three multi-hop QA benchmarks with varying reasoning complexity: **HotpotQA** (Yang et al., 2018) (2-hop), **2WikiMultiHopQA** (Ho et al., 2020) (multi-hop comparison/bridge), and **MuSiQue** (Trivedi et al., 2022) (compositional, 2–4 hops). For each dataset, we use $2{,}000$ training and 500 validation examples and report Exact Match (EM) and token-level F1.

Agent environment. Following prior work (Bo et al., 2024; Tian et al., 2026; Jin et al., 2025), we adopt a multi-turn retrieval-augmented QA environment. At each step, the agent selects `Search[query]` or `Finish[answer]`. A `Search` returns the top-1 passage from a SimCSE retriever (Gao et al., 2021) over the supporting and distractor paragraphs provided with each example, confining retrieval to the dataset context rather than an external corpus. The search budget is 3 for HotpotQA and 2WikiMultiHopQA and 6 for MuSiQue; the environment returns binary EM as $R(\tau) \in \{0, 1\}$. See Algorithm 1 in Appendix A.4 for forced-finish handling and rubric scoring.

Implementation Details. We instantiate ARCO with two comparably sized open-source backbones by default, Qwen3-4B-Instruct-2507 (Yang et al., 2025) and Llama-3.2-3B-Instruct (Grattafiori et al., 2024) (Qwen3-4B and Llama-3.2-3B for short); in each setting, π and μ are initialized from the same backbone and adapted independently with LoRA (Hu et al., 2021), with the generation head producing $K{=}3$ criteria per step unless otherwise specified. For the supervised warmup we use two separate teachers: a policy teacher (`gpt-4o-mini`) rolls out the trajectories, and a rubric teacher (`gpt-5.4`) writes the per-step rubrics and criterion scores. The pairing is deliberate: using a near-perfect policy teacher would leave almost no faulty actions, hence few negative examples for μ to penalize, while the rubric teacher is chosen to provide reliable supervision. RL then follows the sparse-to-dense schedule of §3.2. For each (method, dataset, backbone) cell, we select the post-transition checkpoint with the highest validation EM and report EM/F1 at that checkpoint, using the same selection protocol for every method and ablation to ensure consistent comparison. Dense-stage wall-clock is in Appendix A.5 and full hyperparameters in Appendix A.2.

Baselines. We compare ARCO against representative baselines organized by reward granularity. Together, these baselines cover the three design axes of §2—rubric adaptation, scoring granularity, and judge model—and provide complementary comparisons against the components that distinguish ARCO. We first include a **Base Model** that uses zero-shot prompting without fine-tuning as a reference point, and then compare against two groups of trained baselines. **Group I (Outcome Reward Models, ORMs)**: **Search-R1** (Jin et al., 2025) and **R1-Searcher** (Song et al., 2025) train with binary EM; three rubric-based ORMs score *at the trajectory level* with a frozen closed-source judge but differ in rubric source: **RaR** (Gunjal et al., 2026) pre-generates a query-specific rubric and keeps it fixed; **CARMO** (Gupta et al., 2025) generates context-aware criteria at scoring time conditioned on the query and candidate trajectory; **RLER** (Shao et al., 2025) maintains an on-policy rubric buffer with frozen judge parameters. **Group II (Process Reward Models, PRMs)**: **AgentPRM** (Xi et al., 2026) estimates step values via temporal-difference plus generalized-advantage-estimation (TD+GAE) and is frozen during RL. **ARCO** (ours) generates per-step rubric criteria and predicts rubric-conditioned step scores with an open-source μ matched in scale to π that co-evolves with the policy.

4.2 OVERALL PERFORMANCE (RQ1)

Table 1 compares ARCO with outcome-level RL baselines, rubric-based ORMs, and a process reward model under the same protocol. ARCO achieves the best EM in all six (dataset, backbone) cells

Table 1: Main results on three multi-hop QA benchmarks. All values are reported as percentages, with the "%" symbol omitted for brevity. Rubric-based methods are highlighted in light blue. The best result in each column is shown in **bold**, and the second-best is underlined.

	HotpotQA				2WikiMultiHopQA				MuSiQue			
	Qwen3-4B		Llama-3.2-3B		Qwen3-4B		Llama-3.2-3B		Qwen3-4B		Llama-3.2-3B	
Method	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
Base Model	34.40	45.93	18.40	29.48	47.80	53.95	26.00	32.54	19.40	28.05	9.40	15.12
Group I: Outcome Reward Models (ORM)												
Search-R1 (Jin et al., 2025)	39.40	51.67	34.60	45.54	63.40	67.95	58.20	62.17	24.80	33.66	18.80	26.99
R1-Searcher (Song et al., 2025)	41.00	53.42	35.00	44.48	62.00	67.19	57.80	61.48	26.00	36.02	17.40	26.41
RaR (Gunjal et al., 2026)	38.00	50.75	29.60	40.83	61.40	67.06	53.80	57.88	26.60	**36.46**	16.00	25.26
CARMO (Gupta et al., 2025)	34.80	48.52	30.20	41.74	59.40	65.77	51.60	58.15	22.20	32.26	13.40	21.49
RLER (Shao et al., 2025)	32.20	44.40	28.80	41.31	61.00	66.06	50.00	55.00	24.40	33.35	14.60	23.02
Group II: Process Reward Models (PRM)												
AgentPRM (Xi et al., 2026)	38.60	51.49	29.40	37.29	62.00	65.32	49.60	54.63	25.80	34.81	16.60	25.83
ARCO (ours)	**42.80**	**55.07**	**36.40**	**45.58**	**65.20**	**69.38**	**59.60**	**63.67**	**27.40**	35.55	**21.60**	**29.70**

Table 2: Ablation study. **w/o Rubric** = remove rubric text generation and predict scores directly from trajectory context; **w/o Adaptive** = replace per-step generated rubric with three fixed dataset-level criteria reused across all steps; **w/o Prefix** = let μ generate the rubric per step but condition only on the question and current action, with the trajectory prefix h_t removed from μ's prompt.

	HotpotQA				2WikiMultiHopQA				MuSiQue			
	Qwen3-4B		Llama-3.2-3B		Qwen3-4B		Llama-3.2-3B		Qwen3-4B		Llama-3.2-3B	
Variant	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
SFT Only (π_0)	35.40	47.04	28.00	38.03	60.80	66.07	51.00	54.49	23.20	31.79	12.60	20.72
w/o Rubric	39.80	52.67	35.20	45.16	62.40	67.55	59.00	62.51	26.00	34.58	20.20	28.16
w/o Adaptive	38.00	50.41	35.00	44.56	63.80	68.11	59.20	62.67	26.80	**35.61**	19.80	29.11
w/o Prefix	41.60	54.28	35.40	45.43	64.60	68.64	59.40	63.37	25.60	34.31	20.60	**29.81**
Ours (Full)	**42.80**	**55.07**	**36.40**	**45.58**	**65.20**	**69.38**	**59.60**	**63.67**	**27.40**	35.55	**21.60**	29.70

and the best F1 in five of six. The advantage is thus broad across backbones and metrics rather than concentrated in one setting, with EM gains of $+0.8$ to $+2.8$ points over the strongest competing method and consistent improvements across both backbone families. Among the outcome-level methods, the binary-reward ORMs Search-R1 and R1-Searcher are strong and often outperform the frozen-judge trajectory-level rubric ORMs. ARCO improves over both, with larger EM margins over the rubric ORMs (RaR, CARMO, RLER) than over the binary-reward ones in five of six settings. This is consistent with ARCO's central design choice: adapting the rubric and scorer with the policy at the *step level*, rather than using a fixed outcome-level annotation layer. ARCO also outperforms the step-level PRM AgentPRM by 1.6–10.0 EM across all six cells while providing natural-language evaluation criteria, showing that process-level credit assignment need not rely on opaque scalars. The gain holds on MuSiQue, the most compositional benchmark (2–4 hops), where ARCO leads EM on both backbones ($+0.80$ Qwen, $+2.80$ Llama over the strongest baseline).

4.3 Ablation Study (RQ2)

We ablate three components of ARCO, with results in Table 2. **w/o Rubric** removes generated rubric text and predicts step scores from a mean-pooled trajectory–action representation; **w/o Adaptive** uses a fixed set of dataset-level criteria for every step instead of generating criteria per action; and **w/o Prefix** generates per-step rubrics but conditions μ only on the question and current action, excluding the prior action–observation history. All three improve over SFT. **w/o Rubric** loses EM in all six cells, by up to 3.00 on HotpotQA/Qwen, indicating that rubric text provides semantic structure rather than a post-hoc explanation. **w/o Adaptive** comes closest on 2WikiMultiHopQA/Llama (59.20 vs. 59.60 EM) but still trails by 1.40 EM on 2WikiMultiHopQA/Qwen and by 4.80 EM on HotpotQA/Qwen (38.00 vs. 42.80), showing that fixed criteria do not always capture action-specific failure modes. **w/o**

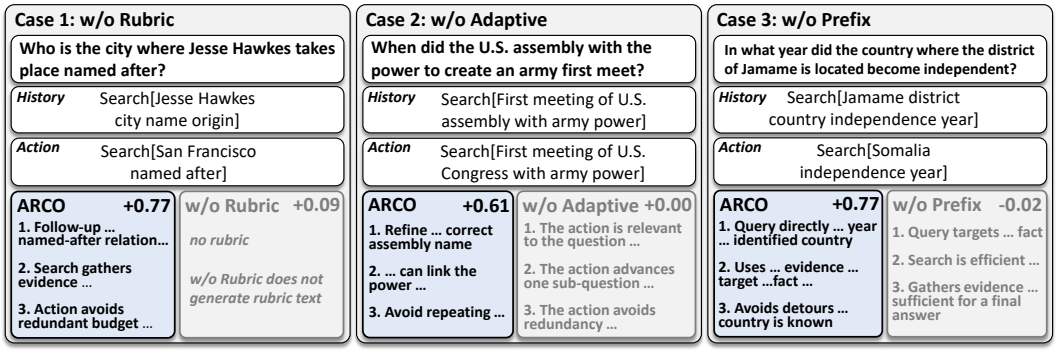


Figure 3: Three single-step comparisons on MuSiQue validation, each contrasting ARCO with one ablation on the same step. Rubric text is abbreviated with ellipses to omit intermediate wording while retaining key phrases relevant to each comparison. ARCO assigns a higher score in each example, while removing rubric text, adaptivity, or prefix information yields a mismatched evaluation.

Table 3: RQ3 step-level rubric quality. **Bind**: rubric-only four-way matching accuracy (%, chance 25.0). **Spec**: step-specificity (1–5). ± are 95% CI half-widths.

		Bind (%, ↑)			**Spec** (1–5, ↑)		
Dataset	Backbone	ALL	Search	Finish	ALL	Search	Finish
HotpotQA	Qwen3-4B	$54.80_{\pm 4.34}$	$50.33_{\pm 5.62}$	$61.50_{\pm 6.68}$	$4.10_{\pm 0.04}$	$4.05_{\pm 0.02}$	$4.17_{\pm 0.09}$
	Llama-3.2-3B	$50.60_{\pm 4.36}$	$43.67_{\pm 5.58}$	$61.00_{\pm 6.70}$	$4.11_{\pm 0.04}$	$4.02_{\pm 0.02}$	$4.25_{\pm 0.08}$
2Wiki	Qwen3-4B	$48.83_{\pm 3.99}$	$46.31_{\pm 4.60}$	$56.21_{\pm 7.76}$	$4.04_{\pm 0.03}$	$4.05_{\pm 0.02}$	$4.01_{\pm 0.09}$
	Llama-3.2-3B	$42.46_{\pm 6.05}$	$36.59_{\pm 7.30}$	$53.41_{\pm 10.21}$	$4.13_{\pm 0.05}$	$4.11_{\pm 0.06}$	$4.17_{\pm 0.10}$
MuSiQue	Qwen3-4B	$44.76_{\pm 6.14}$	$43.92_{\pm 7.90}$	$46.00_{\pm 9.59}$	$4.05_{\pm 0.04}$	$4.01_{\pm 0.06}$	$4.12_{\pm 0.06}$
	Llama-3.2-3B	$43.31_{\pm 6.05}$	$47.95_{\pm 8.00}$	$37.04_{\pm 8.96}$	$3.99_{\pm 0.05}$	$3.97_{\pm 0.05}$	$4.02_{\pm 0.10}$

Prefix is the strongest ablation, slightly improving one F1 cell (MuSiQue/Llama: 29.81 vs. 29.70) but losing EM in every cell, suggesting that prefix conditioning contributes more consistently to exact correctness than to token-level overlap. Figure 3 provides three qualitative examples from MuSiQue validation. Since each ablation trains its own policy, we select matched underlying steps where the question, step index, prior history, and action coincide; each evaluator then receives the inputs prescribed by its variant. The examples illustrate complementary failure modes: removing rubric text eliminates explicit evaluation criteria, fixed criteria remain generic to the current action, and removing the prefix prevents the evaluator from exploiting previously gathered evidence. Appendix C additionally evaluates freezing μ, directly testing the contribution of evaluator co-evolution.

4.4 Rubric Quality: Do Rubrics Identify the Action They Scored? (RQ3)

A rubric about a specific action ought to make that action recoverable. Since, to our knowledge, no existing metric measures whether a rubric is bound to the step it scored, we propose a four-way step-binding diagnostic built from ARCO's validation rollouts: the gold option is the action the rubric was generated for, and the three distractors are unique actions from the same trajectory, so the judge must rely on the rubric text (Appendix D.1 provides sampling and evaluation details). We report two metrics, judged by `gpt-4o-mini`: **Step-Binding Accuracy** (Bind, %), the accuracy of selecting the gold action from the rubric and four candidates alone, with the question and prefix withheld (chance 25%), and **Step-Specificity** (Spec, 1–5), rating how specifically the rubric evaluates the gold action given the question and prefix; both are reported overall and by `Search` and `Finish` steps. Table 3 shows that Bind remains well above the 25% chance level in all six settings, ranging from 42.46% to 54.80%; even the lower bounds of the reported 95% CIs remain above chance, while Spec stays near 4.0. HotpotQA is strongest (54.80% Bind on Qwen), whereas 2Wiki and MuSiQue are harder, as adjacent hops can produce semantically similar `Search` actions for which a specific rubric remains plausible for a nearby distractor. `Finish` Bind exceeds `Search` Bind in five of six cells, consistent with final-answer decisions being easier to identify. The exception is MuSiQue/Llama (37.04 vs. 47.95), on the longest-horizon benchmark, which may reflect forced finishes at budget exhaustion

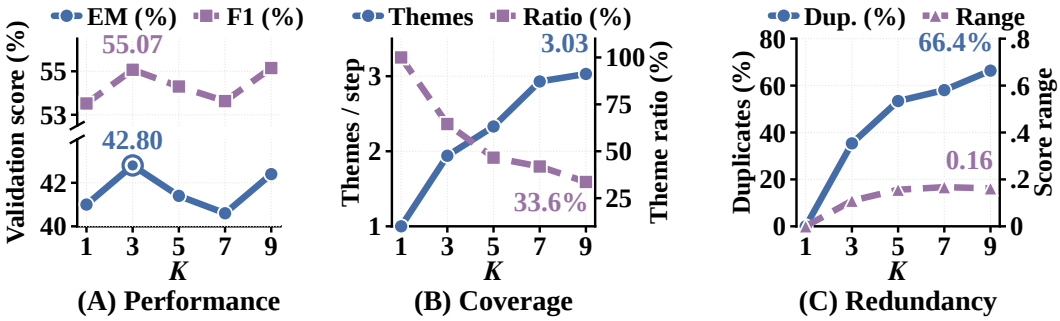


Figure 4: RQ4 K-sensitivity on HotpotQA / Qwen with fixed policy-warmup trajectories and the same training recipe across settings. **(A)** Validation EM/F1 (%) at each best-validation-EM checkpoint (broken y-axis). **(B)** Distinct semantic themes per step (left) and theme ratio $\#\text{themes}/K$ (right, %). **(C)** Total semantic duplicate rate (left, %) and within-cluster max–min score range (right, raw units). (B,C) report validation-step averages, with zero range for steps without duplicates.

whose terminal rubrics resemble generic answer-readiness advice. Appendix D.1 gives successful and failed `Search`/`Finish` examples and directions for more action-discriminative rubrics; Appendix B analyzes how diagnostic the step *scores* are of final success.

4.5 Rubric Width: How Many Criteria Per Step? (RQ4)

The rubric turns ARCO's process reward into a readable step-level signal, and the most direct knob over its shape is the width K—how many criteria μ verbalizes per action. Our main experiments use $K{=}3$; here we ask how this width affects downstream learning. To isolate K, we fix the HotpotQA policy-warmup trajectories and regenerate only their rubric annotations for $K \in \{1, 3, 5, 7, 9\}$; each setting then follows the same μ SFT and RL recipe, with the best-validation-EM checkpoint selected. We diagnose rubric content by embedding each criterion and clustering criteria within every validation step, computing distinct themes, duplicate rate, and within-cluster score range over the resulting clusters. Increasing K does not monotonically improve ARCO (Figure 4A): EM peaks at 42.80 for $K{=}3$, drops at $K{=}5$/7, and only partially recovers at $K{=}9$. Distinct themes saturate quickly: the theme ratio roughly halves between $K{=}3$ and $K{=}9$, so nine criteria still cover about three themes on average (Figure 4B), and duplicates dominate at large K (Figure 4C). These fall into three patterns—*exact duplicates* (byte-identical criteria), *semantic duplicates* (different wordings expressing the same theme), and *diverse* rubrics (criteria spanning distinct themes); Appendix D.2 shows a concrete step of each (Figure 8). Byte-identical duplicates are rare; redundancy at large K is overwhelmingly semantic, while diverse rubrics become the minority as K grows. The within-cluster score range is nonzero but modest on $[-1, 1]$, suggesting that semantic duplicates largely reinforce similar evaluation themes rather than introduce contradictory signals. Thus, increasing K tends to add paraphrased emphasis to existing themes rather than genuinely new dimensions of evaluation.

4.6 Backbone Asymmetry: Rubric Architecture and Scale (RQ5)

Throughout the main experiments, π and μ use the same backbone architecture, coupling rubric-model choice with the policy backbone. RQ5 disentangles them: we fix the policy to Qwen3-4B and vary only the Qwen3-family μ size, then swap backbone families in (π, μ) pairings at comparable scale on HotpotQA. The size sweep (Table 4) is counterintuitive. Policy quality does not improve monotonically with μ capacity: the 0.6B and 4B rubrics tie at 42.80 EM, while the 8B rubric performs worst at 42.00 EM. Within this scale range, increasing rubric-model capacity therefore provides no consistent benefit. One possible explanation is optimization coupling: a higher-capacity μ may reshape the reward landscape more rapidly as π and μ co-evolve, increas-

Table 4: RQ5 results on HotpotQA. Top: μ-size sweep with π fixed to Qwen3-4B. Bottom: cross-family pairs at comparable scale (Qwen3-4B vs. Llama-3.2-3B).

Policy	Rubric	EM	F1
Qwen3-4B	Qwen3-0.6B	**42.80**	54.69
Qwen3-4B	Qwen3-1.7B	42.40	54.54
Qwen3-4B	Qwen3-4B	**42.80**	**55.07**
Qwen3-4B	Qwen3-8B	42.00	54.32
Qwen3-4B	Qwen3-4B	**42.80**	**55.07**
Qwen3-4B	Llama-3.2-3B	41.40	54.44
Llama-3.2-3B	Qwen3-4B	37.20	47.60
Llama-3.2-3B	Llama-3.2-3B	36.40	45.58

ing reward non-stationarity for the policy. These results also suggest that a relatively small μ can suffice in practice. The cross-family comparison probes whether evaluator family matters at comparable scale. For the Qwen policy, replacing the Qwen rubric with Llama lowers EM from 42.80 to 41.40; for the Llama policy, replacing the Llama rubric with Qwen raises EM from 36.40 to 37.20 and F1 from 45.58 to 47.60. ARCO therefore does not require π and μ to share a backbone family, while evaluator architecture can still affect policy learning. Overall, RQ5 suggests that backbone choice matters more than simply scaling μ under our experimental settings.

5 Related Work

Reward Modeling for Multi-Step LLM Agents. RL for multi-step LLM agents typically uses scalar rewards from one of two families. *Outcome reward models* (ORMs) provide trajectory-level supervision and, in agent settings, often train end-to-end from binary task success (Jin et al., 2025; Song et al., 2025; Chen et al., 2026; Zhao et al., 2026; Zeng et al., 2025)—too coarse for step-level credit assignment and potentially vulnerable to reward hacking (Gao et al., 2023). *Process reward models* (PRMs) score individual steps: starting from math step verifiers (Lightman et al., 2024), recent agent variants estimate step values via TD/GAE (Xi et al., 2026; Choudhury, 2025) or through tree search, step-wise PPO, and implicit step supervision (Xiong et al., 2024; Wang et al., 2025a;b; Liu et al., 2026b). Most produce opaque scalars that do not explain why a step is good or bad. A newer line makes step-level feedback interpretable via checklists (Chae et al., 2026), principle-guided reasoning (Zhang et al., 2026b), or implicit step rewards for multi-hop QA from outcome supervision alone (Wang et al., 2026a); these methods keep the evaluator fixed during policy optimization, whereas ARCO continues updating its evaluator on the policy's own rollouts.

Rubric-Based Rewards and Joint Policy–Reward Learning. To improve interpretability, *rubric-based rewards* ground evaluation in natural-language criteria. RaR (Gunjal et al., 2026) freezes a query-specific rubric before RL; CARMO (Gupta et al., 2025) generates rubrics at scoring time but freezes the generator; others focus on synthesizing or eliciting rubrics (Liu et al., 2026a; Huang et al., 2025b; Zhang et al., 2026a; Rezaei et al., 2025); RLER (Shao et al., 2025) evolves a rubric buffer during training; and RubricARM (Xu et al., 2026) adds alternating training. Most of these methods score *at the trajectory level*, often relying on a frozen closed-source judge. A separate thread *jointly trains the policy and reward model* (Hong et al., 2025; Shi et al., 2025; Huang et al., 2025a; Wang et al., 2026b;c), but uses opaque scalars or lacks step-level supervision. **ARCO** brings these axes together: it generates *step-level* rubrics with an open-source model and jointly adapts the rubric model and its scoring function with the policy on on-policy rollouts, without an external judge.

6 Limitations

ARCO's process signal is a per-step scalar tied to the outcome through a sum constraint, which lacks positional structure and leaves early `Search` scores less diagnostic than terminal ones (Appendix B). Position-aware supervision with explicit temporal structure could further improve credit assignment. The pace of π–μ co-evolution is set by learning rates and regularization weights; adaptive update ratios and drift-aware scheduling remain future work. Training both models and invoking μ at every step costs more than single-model RL, especially at long horizons; distillation, rubric caching, or asynchronous μ updates may reduce this cost. Finally, we evaluate only on multi-hop QA and assess rubric quality with our own diagnostics rather than expert-annotated evaluator benchmarks.

7 Conclusion

We presented ARCO, an adaptive rubric co-evolution framework for multi-step LLM agents. A trainable rubric model shares a backbone across rubric generation and scoring while remaining separate from the policy; both co-evolve on shared on-policy rollouts. After warmup, terminal outcomes supervise rubric-conditioned step rewards without an external judge, allowing the evaluator to track policy changes and reshape rewards. Across three multi-hop QA benchmarks and two backbone families, ARCO achieves the highest EM among evaluated methods while providing readable evaluation criteria. Rubric diagnostics show how these criteria distinguish actions and where ambiguity and semantic redundancy remain, motivating better learned evaluators.

AI USE STATEMENT

We used generative AI tools to assist with literature research, translation and language editing, coding and feedback on experimental analysis and interpretation of results, figure and caption preparation, and manuscript formatting. We additionally used GPT models to generate and annotate warmup trajectories and step-level rubrics for supervised training (Appendix A) and to assist with automated rubric-quality evaluations (Section 4.4). All AI-assisted outputs were reviewed and verified by the authors. The authors made the final methodological, experimental, and interpretive decisions and take responsibility for the final paper, including its claims, analyses, code, and artifacts.

ETHICS STATEMENT

This work uses publicly released benchmarks (HotpotQA, 2WikiMultiHopQA, and MuSiQue) and publicly released models (Qwen3-4B-Instruct-2507 and Llama-3.2-3B-Instruct) under their respective licenses. Our study involves no human-subject experiments or collection of private personal data. We use GPT APIs for warmup trajectory generation, rubric annotation, and the RQ3 automated evaluations; ARCO's online reward computation uses its own rubric model. ARCO studies interpretable process-reward learning for multi-step language agents. Its generated criteria and scores may inherit model biases and factual errors, and their readability should not be taken as a guarantee of correctness or suitability for consequential decisions without further human verification or oversight.

REPRODUCIBILITY STATEMENT

We have made an effort to make ARCO reproducible. The full method is specified in Section 3, with the formal training loop in Algorithm 1 (Appendix A.4). Section 4 describes the agent environment, datasets, evaluation protocol, and checkpoint-selection rule. Appendix A details warmup trajectory collection, rubric-score projection, the dual SFT objective, and online training, with shared and per-setting hyperparameters in Table 5. Appendix A.5 reports ARCO's dense-stage wall-clock times to the best post-transition checkpoints by validation EM. The policy, step-level rubric, trajectory-level rubric, and RQ3 judge prompt templates are provided in Appendix E. Source code, training scripts, configuration files, and the warmup data used in the main-table HotpotQA / Qwen run are available at `https://anonymous.4open.science/r/arco-anon-2C61`.

REFERENCES

Xiaohe Bo, Zeyu Zhang, Quanyu Dai, Xueyang Feng, Lei Wang, Rui Li, Xu Chen, and Ji-Rong Wen. Reflective multi-agent collaboration based on large language models. *Advances in Neural Information Processing Systems*, 37:138595–138631, 2024.

Hyungjoo Chae, Seonghwan Kim, Junhee Cho, Seungone Kim, Seungjun Moon, Gyeom Hwangbo, Dongha Lim, Minjin Kim, Yeonjun Hwang, Minju Gwak, et al. Web-shepherd: Advancing prms for reinforcing web agents. *Advances in Neural Information Processing Systems*, 38:63314–63356, 2026.

Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Chenzheng Zhu, Haofen Wang, Jeff Pan, Wen Zhang, Huajun Chen, Fan Yang, et al. Learning to reason with search for llms via reinforcement learning. *Advances in Neural Information Processing Systems*, 38:85287–85307, 2026.

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

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International conference on machine learning*, pp. 10835–10866. PMLR, 2023.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 6894–6910, 2021.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. In *International Conference on Learning Representations*, volume 2026, pp. 127924–127945, 2026.

Taneesh Gupta, Shivam Shandilya, Xuchao Zhang, Rahul Madhavan, Supriyo Ghosh, Chetan Bansal, Huaxiu Yao, and Saravan Rajmohan. Carmo: Dynamic criteria generation for context aware reward modelling. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 2202–2261, 2025.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Haitao Hong, Yuchen Yan, Xingyu Wu, Guiyang Hou, Wenqi Zhang, Weiming Lu, Yongliang Shen, and Jun Xiao. Cooper: Co-optimizing policy and reward models in reinforcement learning for large language models. *arXiv preprint arXiv:2508.05613*, 2025.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Changxin Huang, Yanbin Chang, Junfan Lin, Junyang Liang, Runhao Zeng, and Jianqiang Li. Efficient language-instructed skill acquisition via reward-policy co-evolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 14576–14584, 2025a.

Zenan Huang, Yihong Zhuang, Guoshan Lu, Zeyu Qin, Haokai Xu, Tianyu Zhao, Ru Peng, Jiaqi Hu, Zhanming Shen, Xiaomeng Hu, et al. Reinforcement learning with rubric anchors. *arXiv preprint arXiv:2508.12790*, 2025b.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Vivek Kalyan and Martin Andrews. Reinforcement learning for long-horizon multi-turn search agents. *arXiv preprint arXiv:2510.24126*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Tianci Liu, Ran Xu, Tony Yu, Ilgee Hong, Carl Yang, Tuo Zhao, and Haoyu Wang. Openrubrics: Towards scalable synthetic rubric generation for reward modeling and llm alignment. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17417–17437, 2026a.

Xiaoqian Liu, Ke Wang, Yuchuan Wu, Fei Huang, Yongbin Li, Jianbin Jiao, and Junge Zhang. Agentic reinforcement learning with implicit step rewards. In *International Conference on Learning Representations*, volume 2026, pp. 129271–129291, 2026b.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

MohammadHossein Rezaei, Robert Vacareanu, Zihao Wang, Clinton Wang, Bing Liu, Yunzhong He, and Afra Feyza Akyürek. Online rubrics elicitation from pairwise comparisons. *arXiv preprint arXiv:2510.07284*, 2025.

Rulin Shao, Akari Asai, Shannon Zejiang Shen, Hamish Ivison, Varsha Kishore, Jingming Zhuo, Xinran Zhao, Molly Park, Samuel G Finlayson, David Sontag, et al. Dr tulu: Reinforcement learning with evolving rubrics for deep research. *arXiv preprint arXiv:2511.19399*, 2025.

Tianyuan Shi, Canbin Huang, Fanqi Wan, Longguang Zhong, Ziyi Yang, Weizhou Shen, Xiaojun Quan, and Ming Yan. Mutual-taught for co-adapting policy and reward models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16285–16298, 2025.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025.

Zihang Tian, Rui Li, Jingsen Zhang, Xiaohe Bo, Wei Huo, and Xu Chen. Haps: Hierarchical llm routing with joint architecture and parameter search. *arXiv preprint arXiv:2601.05903*, 2026.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition, 2022. URL https://arxiv.org/abs/2108.00573.

Xinyi Wang, Yiping Song, Zhiliang Tian, Bo Liu, Tingjin Luo, and Minlie Huang. Dprm: A dual implicit process reward model in multi-hop question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33683–33691, 2026a.

Xinyu Wang, Hanwei Wu, Jingwei Song, Shuyuan Zhang, Jiayi Zhang, Fanqi Kong, Tung Sum Thomas Kwok, Xiao-Wen Chang, Yuyu Luo, Chenglin Wu, et al. Co-evolution of policy and internal reward for language agents. *arXiv preprint arXiv:2604.03098*, 2026b.

Yinjie Wang, Tianbao Xie, Ke Shen, Mengdi Wang, and Ling Yang. Rlanything: Forge environment, policy, and reward model in completely dynamic rl system. *arXiv preprint arXiv:2602.02488*, 2026c.

Ziliang Wang, Xuhui Zheng, Kang An, Cijun Ouyang, Jialu Cai, Yuhang Wang, and Yichao Wu. Stepsearch: Igniting llms search ability via step-wise proximal policy optimization. *arXiv preprint arXiv:2505.15107*, 2025a.

Zilong Wang, Jingfeng Yang, Sreyashi Nag, Samarth Varshney, Xianfeng Tang, Haoming Jiang, Jingbo Shang, and Sheikh Muhammad Sarwar. Rro: Llm agent optimization through rising reward trajectories. *arXiv preprint arXiv:2505.20737*, 2025b.

Zhiheng Xi, Chenyang Liao, Guanyu Li, Zhihao Zhang, Wenxiang Chen, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, Wei Wu, et al. Agentprm: Process reward models for llm agents via step-wise promise and progress. In *Proceedings of the ACM Web Conference 2026*, pp. 4184–4195, 2026.

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, 2024.

Ran Xu, Tianci Liu, Zihan Dong, Tony Yu, Ilgee Hong, Carl Yang, Linjun Zhang, Tao Zhao, and Haoyu Wang. Alternating reinforcement learning for rubric-based reward modeling in non-verifiable llm post-training. *arXiv preprint arXiv:2602.01511*, 2026.

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

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William 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, 2018.

Siliang Zeng, Quan Wei, William Brown, Oana Frunza, Yuriy Nevmyvaka, Yang Katie Zhao, and Mingyi Hong. Reinforcing multi-turn reasoning in llm agents via turn-level credit assignment. In *ICML 2025 Workshop on Computer Use Agents*, 2025.

Junkai Zhang, Zihao Wang, Lin Gui, Swarnashree Mysore Sathyendra, Jaehwan Jeong, Victor Veitch, Wei Wang, Yunzhong He, Bing Liu, and Lifeng Jin. Chasing the tail: Effective rubric-based reward modeling for large language model post-training. In *International Conference on Learning Representations*, volume 2026, pp. 133430–133457, 2026a.

Yao Zhang, Shijie Tang, Zeyu Li, Zhen Han, and Volker Tresp. Webarbiter: A generative reasoning process reward model for web agents. In *The Fourteenth International Conference on Learning Representations*, 2026b.

Qingfei Zhao, Ruobing Wang, Dingling Xu, Daren Zha, Ma Bowen, Zhichun Wang, Shijie Jia, Limin Liu, and Xin Wang. R-search: Empowering llm reasoning with search via multi-reward reinforcement learning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 38030–38046, 2026.

A TRAINING DETAILS

This appendix details ARCO's two training stages: the supervised warmup that initializes π and μ, and the full on-policy co-evolution loop.

A.1 WARMUP STAGE

ARCO uses a supervised warmup before on-policy co-evolution so that both the policy and the rubric model start from a working initialization. We collect warmup records with two separate teachers: a weaker policy teacher (`gpt-4o-mini`) rolls out trajectories on the training questions in the same retrieval environment used for RL, and a stronger rubric teacher (`gpt-5.4`) annotates each complete trajectory with per-step criteria and scores. We deliberately pair a weaker policy teacher with a stronger rubric teacher: a near-perfect policy teacher would produce almost no faulty actions, leaving the rubric model with no negative examples to learn to penalize, while the stronger rubric teacher keeps the annotations reliable. For each question we sample multiple trajectories and log every action, observation, parse status, search budget, and terminal reward. HotpotQA and 2Wiki use a search budget of three, while MuSiQue uses six; the terminal reward is the task EM reward used by the main experiments throughout training and evaluation.

Policy Warmup. The policy SFT dataset is built only from the correct trajectories (those that end with EM=1), converted into one supervised example per step. The input contains the question, the trajectory prefix before the current action, and the remaining search budget; when the budget is exhausted and the dataset supports forced finishing, the prompt switches to a finish-only template. The target is the recorded action string, e.g., `Search[query]` or `Finish[answer]`, which initializes π to imitate successful tool-use traces rather than all sampled behavior during supervised initialization.

Rubric Warmup. The rubric annotator sees the full sampled trajectory and returns, for each step, K natural-language criteria and K criterion scores. Scores are parsed, clipped to $[-1, 1]$, padded or truncated to width K, and projected so that the summed step-score means satisfy the trajectory-level decomposition constraint,

$$\sum_t \frac{1}{K} \sum_{j=1}^{K} \hat{s}_{t,j} = R(\tau), \tag{5}$$

where $\hat{s}_{t,j}$ is the projected score of criterion j at step t and the vector $\hat{\mathbf{s}}_t = (\hat{s}_{t,1}, \ldots, \hat{s}_{t,K})$ collects them for Eq. 7. Although the teacher sees the complete trajectory for offline annotation, μ is trained only on the pre-action prefix and current action, matching its online inputs. This preserves the same information boundary used when μ scores on-policy actions.

Concretely, let $v_1, \ldots, v_{KT}$ be the clipped teacher scores flattened over all T steps and K criteria. Since each step score is the mean of its K criteria, Eq. 5 is equivalent to the single linear constraint $\sum_i \hat{s}_i = K\,R(\tau)$. We take the Euclidean projection of v onto the intersection of that hyperplane with the box $[-1, 1]^{KT}$,

$$\hat{s} = \arg\min_x \|x - v\|_2^2 \quad \text{s.t.} \quad \textstyle\sum_i x_i = K\,R(\tau), \;\; x \in [-1, 1]^{KT}, \tag{6}$$

whose solution has the closed form $\hat{s}_i = \mathrm{clamp}(v_i + \lambda, -1, 1)$ for a single scalar λ; we recover λ by bisection (80 iterations, bracket $[-2, 2]$), which suffices because $\sum_i \mathrm{clamp}(v_i + \lambda, -1, 1)$ is non-decreasing in λ. The target total is clipped to the reachable range $[-KT, KT]$ before solving, and any residual above numerical tolerance is distributed over coordinates still inside the box. Being an ℓ_2 projection, the operation makes the warmup data compatible with the later decomposition loss while perturbing the teacher's scores as little as possible, so their relative ordering is largely preserved. The trajectory-level prompt is shown in Appendix E. Before μ SFT, records are filtered by the same constraint with a small tolerance, and steps with empty criteria are dropped.

The rubric model μ is then trained with a dual objective: the generation head learns to generate the criterion JSON for each state-action pair, while the score head regresses the annotated criterion scores from the mean-pooled hidden representation of the full scoring sequence. The SFT loss is

$$\mathcal{L}_\mu^{\text{sft}} = \underbrace{-\sum_t \log p_\theta(\tilde{r}_t \mid h_t, a_t)}_{\text{rubric text loss}} + \lambda_{\text{sft}} \underbrace{\sum_t \frac{1}{K} \|\mathbf{d}_t - \hat{\mathbf{s}}_t\|_2^2}_{\text{criterion-score loss}} \tag{7}$$

where $\tilde{r}_t$ is the annotated criterion list, $\hat{\mathbf{s}}_t \in [-1,1]^K$ the projected score vector, $\mathbf{d}_t$ the score-head prediction, and λ_{sft} balances the two terms (distinct from the RL-stage KL weights λ_π, λ_μ of §3.2). The default $K{=}3$; ablations reuse the same pipeline but either remove the rubric text target or replace it with a fixed rubric.

After warmup, the selected policy checkpoint initializes the evolving π and the frozen reference π_{ref}, and the selected rubric checkpoint initializes the evolving μ and the frozen reference μ_{ref}; both references anchor the KL terms during RL.

A.2 Hyperparameters

Table 5 lists every hyperparameter needed to reproduce the main-table ARCO runs. Although π and μ are initialized from the same base checkpoint, they are *separate model instances with their own LoRA adapters* (held on different devices in our implementation); no adapter weights are shared between them, so the policy-gradient and decomposition gradients never write to the same parameters. The rubric-generation branch and the score head do share μ's backbone adapter, which is the coupling discussed in §3.1. All runs use a single seed (42); the shared settings below are identical across all six cells, and only the per-cell block varies.

Table 5: ARCO hyperparameters. *Top*: settings shared by all six (dataset, backbone) cells. *Bottom*: per-cell settings. λ_{sft} is the criterion-score loss weight of Eq. 7; α is the dense-reward scale of §3.2; η is the number of sparse epochs.

Shared across all cells	
LoRA rank / α_{LoRA} / dropout / targets	8 / 32 / 0.05 / `q_proj`, `v_proj`
π SFT: epochs / lr / batch $\times$ accum	6 / $1{\times}10^{-5}$ / 2×32
μ SFT: epochs / lr / score-head lr / batch $\times$ accum	10 / $5{\times}10^{-5}$ / $1{\times}10^{-3}$ / 4×8
μ SFT: λ_{sft} / LR warmup ratio / weight decay	10.0 / 0.03 / 0.01
RL: weight decay / sparse rollout group size / score mini-batch	0.01 / 4 / 8
RL: max new tokens (π / μ)	64 / 128
Criteria per step K / seed / max sequence length	3 / 42 / 2048
Per cell, ordered HotpotQA | 2Wiki | MuSiQue, each as Qwen3-4B | Llama-3.2-3B	
RL epochs	8 | 10 | 10 | 10 | 10 | 10
Sparse epochs η	3 | 3 | 4 | 4 | 3 | 3
Batch size	64 | 64 | 64 | 64 | 48 | 64
Rollout temperature	0.7 | 0.7 | 0.8 | 0.7 | 0.7 | 0.7
Sparse-stage π lr ($\times 10^{-6}$)	10 | 5 | 10 | 10 | 5 | 10
Dense-stage π lr ($\times 10^{-6}$)	3 | 1 | 2 | 0.5 | 1 | 5
μ lr ($\times 10^{-5}$)	1 | 3 | 1 | 3 | 1 | 1
Score-head lr ($\times 10^{-4}$)	5 | 10 | 1 | 10 | 0.3 | 1
λ_π (dense-stage π KL)	0.05 | 0.15 | 0.15 | 0.25 | 0.22 | 0.22
λ_μ (μ KL)	0.05 | 0.05 | 0.10 | 0.05 | 0.15 | 0.10
Dense-reward scale α	0.5 | 0.6 | 0.5 | 0.2 | 1.2 | 1.0
Search budget B	3 | 3 | 3 | 3 | 6 | 6

The λ_π row gives the dense-stage policy KL weight, which applies from epoch $\eta + 1$ onward together with the dense-stage learning rate, counting epochs from one. The configuration field `dense_start_epoch` uses a zero-based index and equals η; for example, $\eta = 3$ means three sparse epochs followed by dense training from epoch four. On HotpotQA the sparse stage runs with no policy KL ($\lambda_\pi{=}0$), since the policy is still close to its SFT initialization there; on the other four cells the same λ_π is used in both stages.

A.3 RL Objective and Baselines

For Eq. 4, let L_t be the number of generated rubric tokens at step t, and let $p_{\theta,t,i}$ and $p_{\theta_{\text{ref}},t,i}$ be the current and frozen-reference next-token distributions, both conditioned on (h_t, a_t) and the same

sampled rubric prefix $r_{t,<i}$. The rubric regularizer is

$$\mathcal{K}_\tau = \sum_t \frac{1}{L_t} \sum_{i=1}^{L_t} D_{\mathrm{KL}}(p_{\theta_{\mathrm{ref}},t,i} \,\|\, p_{\theta,t,i}) \,. \tag{8}$$

Thus we average KL over tokens within each step and sum over steps; steps with no generated rubric tokens contribute zero to this term. The reference distributions are held fixed during optimization. The policy KL in Eq. 3 uses the same reference-to-current direction and token averaging, evaluated on the sampled action prefixes, summed over valid policy steps and averaged over trajectories.

The decomposition loss is balanced by outcome class within each trajectory batch. For $c \in \{0, 1\}$, define $n_c = \sum_{\tau \in \mathcal{B}} \mathbf{1}[R(\tau) = c]$. The weight in Eq. 4 is

$$w_\tau = \frac{|\mathcal{B}|}{2 \max\{n_{R(\tau)}, 1\}}. \tag{9}$$

This weight applies only to the squared decomposition error; the KL term is not class-weighted. Both model updates average gradients over the trajectory batch.

During sparse training, we sample four rollouts per question and subtract the mean outcome within that question's rollout group. During dense training, we sample one rollout per question and first compute every step's reward-to-go and position bucket, then the bucket means, and finally the advantages. Question batches are sized so that the resulting trajectory batch has the configured batch size. The dense baseline includes the step being centered; a bucket containing only one step falls back to the global mean reward-to-go across the batch. Search steps are grouped by search rank, while non-search actions and the final step are assigned to the terminal bucket.

A.4 Full Training Algorithm

Algorithm 1 details the full ARCO training loop. We distinguish the question batch $\mathcal{Q}_b$ from the resulting trajectory batch $\mathcal{B}$. The routine EnvStep returns an observation, the remaining budget, a termination flag, and the outcome reward (zero before termination). A valid `Search` retrieves a passage and consumes one budget unit; a malformed action returns an error observation and also consumes one unit. A `Finish` terminates with the answer's EM reward. After budget exhaustion, one additional action is generated with the finish-only prompt: any non-`Finish` response terminates with zero reward and a violation observation, without executing a search. This terminal step is still scored by μ. Rubrics always use the prefix before the current action, and stored rollouts are shared by the two separate model updates.

A.5 Dense-Stage Training Time

Table 6 reports the wall-clock time in the dense-stage up to the best dense-stage checkpoint by EM validation.

B Where Does the Step-Level Signal Live?

Section 4.4 shows that ARCO's rubrics are bound to the actions they score. Here we ask a complementary quantitative question about the *scores* rather than the rubric text: on which actions does the step score separate trajectories that end correct from those that end wrong? For each (dataset, backbone) cell, we use the saved validation trace from the best-validation-EM checkpoint, requiring no additional training or inference. Splitting steps into `Search` and `Finish`, we measure two quantities: the **Gap**, the mean step score on trajectories ending correct (EM=1) minus that on trajectories ending wrong; and the **AUROC** of the step score as a predictor of final correctness (chance 0.5, with higher values indicating stronger separation).

Table 7 reveals a consistent asymmetry between the two action types across all six cells. The `Finish` score is strongly diagnostic of final success (Gap 0.33–0.44, AUROC 0.82–0.87), whereas the `Search` score is substantially less so (Gap 0.03–0.08, AUROC 0.58–0.65). Thus, terminal scores are much more predictive of the final outcome than scores assigned to intermediate retrieval decisions.

Algorithm 1 ARCO: Adaptive Rubric Co-Evolution

Require: SFT-initialized π, μ; frozen references $\pi_{\text{ref}}, \mu_{\text{ref}}$; training questions $\mathcal{E}$; retriever $\mathcal{R}$; search budget B; epochs E; sparse epochs η

for $e = 1, \ldots, E$ **do**
  use_dense $\leftarrow (e > \eta)$; $m \leftarrow 1$ if dense, otherwise 4
  for question batch $\mathcal{Q}_b \subset \mathcal{E}$ **do**
    $\mathcal{B} \leftarrow \varnothing$ ▷ Trajectory batch
    for each $(q, j) \in \mathcal{Q}_b \times \{1, \ldots, m\}$ **do**
      $h_1 \leftarrow q$; $b \leftarrow B$; $t \leftarrow 1$; $\tau \leftarrow [\,]$; *done* $\leftarrow$ false
      while not done **do**
        if $b = 0$ **then**
          $a_t \sim \pi(\cdot \mid h_t, \textsc{FinishOnly})$
        else
          $a_t \sim \pi(\cdot \mid h_t)$
        end if
        $(o_t, b', \textit{done}, R) \leftarrow \textsc{EnvStep}(q, a_t, b, \mathcal{R})$
        $r_t \sim p_\theta(\cdot \mid h_t, a_t)$ ▷ Rubric from the pre-action prefix
        $\bar{\mathbf{h}}_t \leftarrow \mathrm{MeanPool}(f_\theta(h_t, a_t, r_t))$
        $\mathbf{d}_t \leftarrow g_\phi(\bar{\mathbf{h}}_t)$; $s_t \leftarrow \mathrm{mean}(\mathbf{d}_t)$
        Append $(h_t, a_t, o_t, r_t, s_t)$ and action-validity flag to τ
        $h_{t+1} \leftarrow (h_t, a_t, o_t)$; $b \leftarrow b'$; $t \leftarrow t + 1$
      end while
      Store (τ, R, q) in $\mathcal{B}$
    end for
    if use_dense **then**
      Compute $G_{\tau,t} \leftarrow \sum_{k \geq t} s_{\tau,k}$ and $b_{\tau,t} \leftarrow \mathrm{Bucket}(\tau, t)$ for all steps
      Compute $\bar{G}_b \leftarrow \mathrm{mean}\{G_{\tau,t} : b_{\tau,t} = b\}$ for each bucket
      For singleton buckets, replace $\bar{G}_b$ by the global step mean
      Set $A_{\tau,t} \leftarrow \alpha(G_{\tau,t} - \bar{G}_{b_{\tau,t}})$ for all steps
    else
      Compute $\bar{R}_q \leftarrow \mathrm{mean}\{R(\tau) : \tau \in \mathcal{B}$ belongs to $q\}$ for each question
      Set $A_{\tau,t} \leftarrow R(\tau) - \bar{R}_q$ for every step of each trajectory
    end if
    Compute outcome-class weights w_τ (Appendix A.3)
    Update π using valid policy steps and Eq. 3
    Update μ on the same $\mathcal{B}$ using Eq. 4
  end for
end for

This asymmetry is consistent with the single-form process signal noted in our Limitations (§6). The only trajectory-level supervision on the scores is the decomposition constraint $\sum_t s_t = R(\tau)$ (Eq. 4), which constrains their sum but does not specify how outcome credit should be distributed across steps. A `Finish` action directly contains the submitted answer, making its quality comparatively easy to relate to the terminal outcome. By contrast, the utility of a `Search` action often becomes apparent only through later retrieval and answering steps, while the decomposition constraint provides no explicit temporal credit structure. The measured Gap and AUROC are consistent with this interpretation: score quality is strongest at the terminal decision and weakest on the earlier retrieval decisions where process-level credit assignment is most needed. This suggests a concrete direction for future work: replacing the uniform sum constraint with position-aware or discounted decomposition to push more diagnostic signal toward earlier steps.

C Exploratory Study: Freezing the Rubric Model

The ablations in §4.3 vary what the rubric model *sees* and *writes*, but all allow μ to adapt during RL. We therefore run an exploratory **w/o Co-evolution** variant that removes this adaptation entirely. After supervised warmup, its rubric generator and score head remain frozen throughout both the sparse and dense RL stages: μ receives no on-policy updates and uses no μ KL term. By contrast, ARCO updates μ during sparse training and continues co-evolving it with π during dense training.

Table 6: Dense-stage training time to the best-validation-EM dense-stage checkpoint. EM/F1 are percentages with the % symbol omitted.

Setting	Variant	EM / F1	Dense time
Main-table ARCO runs			
HotpotQA	Qwen3-4B	42.80 / 55.07	1h 22m
HotpotQA	Llama-3.2-3B	36.40 / 45.58	2h 38m
2WikiMultiHopQA	Qwen3-4B	65.20 / 69.38	5h 48m
2WikiMultiHopQA	Llama-3.2-3B	59.60 / 63.67	1h 52m
MuSiQue	Qwen3-4B	27.40 / 35.55	15h 29m
MuSiQue	Llama-3.2-3B	21.60 / 29.70	12h 49m
RQ4 rubric-width sweep on HotpotQA / Qwen			
K=1	Qwen3-4B	41.00 / 53.52	2h 19m
K=3	Qwen3-4B	42.80 / 55.07	1h 22m
K=5	Qwen3-4B	41.40 / 54.30	1h 20m
K=7	Qwen3-4B	40.60 / 53.63	1h 20m
K=9	Qwen3-4B	42.40 / 55.15	1h 23m
RQ5 Qwen-family rubric-size sweep on HotpotQA, fixed π=Qwen3-4B			
μ=Qwen3-0.6B	Qwen3-4B	42.80 / 54.69	1h 25m
μ=Qwen3-1.7B	Qwen3-4B	42.40 / 54.54	1h 27m
μ=Qwen3-4B	Qwen3-4B	42.80 / 55.07	1h 22m
μ=Qwen3-8B	Qwen3-4B	42.00 / 54.32	2h 39m

Table 7: Discriminativeness of the ARCO step score at the best-validation-EM checkpoint for each cell. Gap is the mean step score on trajectories that end correct (EM=1) minus that on wrong ones; AUROC uses the step score to rank correct vs. wrong trajectories (chance 0.5). Finish actions carry a far stronger signal than Search actions in every cell.

		Search		**Finish**	
Dataset	Backbone	Gap	AUROC	Gap	AUROC
HotpotQA	Qwen3-4B	0.071	0.604	0.329	0.816
HotpotQA	Llama-3.2-3B	0.049	0.650	0.420	0.827
2Wiki	Qwen3-4B	0.082	0.642	0.331	0.826
2Wiki	Llama-3.2-3B	0.079	0.647	0.441	0.873
MuSiQue	Qwen3-4B	0.078	0.583	0.352	0.844
MuSiQue	Llama-3.2-3B	0.030	0.600	0.327	0.844

For the comparison in Figure 5, both dense runs start from the same post-sparse policy checkpoint, so π has an identical initialization when the plotted curves begin. The difference lies in the evaluator: ARCO enters the dense stage with a μ already adapted on sparse-stage on-policy rollouts and continues updating it, whereas **w/o Co-evolution** uses only the warmup-SFT μ and keeps it fixed throughout RL. The dense-stage configurations otherwise match in batch size, rollout temperature, policy KL weight (λ_π=0.08), policy learning rate (2.5×10^{-6}), and dense-reward scale. The co-evolving and frozen runs use 6 and 8 dense epochs, respectively; neither is stopped early.

The two policies behave similarly at the beginning of dense training, as expected from their shared policy initialization, but their trajectories then diverge. The co-evolving-μ run recovers from its early dip and reaches 36.40 EM, whereas the frozen-μ run peaks at 35.20 in its first dense epoch and declines in every subsequent epoch, ending at 28.00 EM. Giving the frozen evaluator two additional dense epochs does not reverse this trend; instead, the gap continues to widen.

This pattern is consistent with evaluator–policy mismatch growing over training. The frozen μ has seen only warmup data and never adapts to the policy's on-policy behavior during sparse or dense RL.

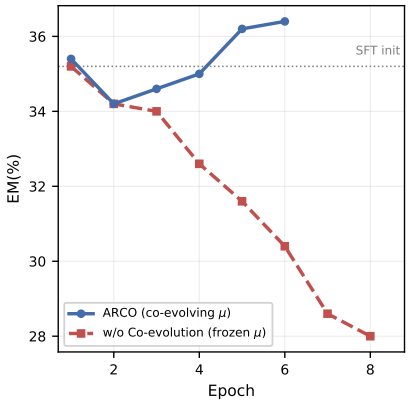


Figure 5: Validation EM during dense training on HotpotQA/Llama-3.2-3B. Policies start from the same post-sparse checkpoint; the co-evolving μ recovers, whereas the frozen μ continues to decline.

As π continues to change, its states and actions can move further from the distribution on which the frozen evaluator was calibrated, making its rubric-conditioned scores progressively less informative for the current policy. ARCO instead updates μ on the policy's own rollouts, allowing the evaluator to track this distribution shift. The experiment does not by itself prove that evaluator staleness causes the decline, but the widening gap is consistent with the motivation for co-evolution in §3.2.

We stress that this is an exploratory comparison rather than a main ablation. We plot the HotpotQA/Llama setting for which the dense-stage policy initialization and optimization configuration are directly matched. On 2WikiMultiHopQA with the Llama backbone, the frozen-μ run shows a similar decline, from 57.00 EM in its first dense epoch to 51.60 by its eighth. On the two Qwen3-4B settings, by contrast, the frozen-μ curves remain approximately flat over a comparable horizon rather than degrading. We did not run this variant on MuSiQue. These results suggest that evaluator staleness can be harmful in some settings but is not uniformly so; establishing when co-evolution is necessary would require a broader matched sweep.

D Additional Qualitative Examples

D.1 Rubric Binding (RQ3)

Distractor Sampling. The four-way binding protocol of §4.4 draws all three distractors from the *same* trajectory as the gold action, which limits how many steps are usable. With a search budget of 3 (HotpotQA and 2WikiMultiHopQA), a trajectory contains at most four actions, and repeated queries reduce the number of *unique* actions further, so a trajectory must contain at least four distinct actions for any of its steps to be scorable. We therefore restrict the evaluated population to steps from trajectories meeting that condition rather than padding with weaker negatives. Table 8 reports the resulting coverage: 67.0% of validation steps qualify overall, ranging from 46.4% (2Wiki/Qwen) to 78.2% (MuSiQue/Qwen), with the longer-horizon MuSiQue cells retaining the most.

Sampling and Valid Responses. For 2Wiki/Qwen we sample 600 eligible steps (447 `Search`, 153 `Finish`). For each of the other five settings we sample 300 `Search` and 200 `Finish` steps, using seed 42. Each step receives separate Bind and Spec requests to `gpt-4o-mini`. We compute each metric and its confidence interval from valid responses to that request type; API errors and invalid responses are excluded rather than counted as incorrect or assigned a score. Server errors and rate limits left fewer valid responses in three settings, as reported in Table 9. Thus the sample sizes differ across settings and between Bind and Spec. The ALL columns in Table 3 summarize the valid sampled responses without reweighting to the action-type distribution of the validation set.

The trajectory-length restriction is not random: short trajectories often arise when a small number of well-targeted searches already suffices, whereas the binding protocol requires at least four distinct actions. The Bind results in Table 3 should therefore be interpreted as conditional on this longer-trajectory subset rather than as estimates over all validation steps. Shorter trajectories also provide

Table 8: Coverage of the RQ3 binding diagnostic. A trajectory is eligible only if it contains ≥ 4 unique actions, so that three same-trajectory distractors exist. Fractions and percentages are relative to all validation trajectories or rubric-bearing validation steps, respectively, at each cell's best-validation-EM checkpoint.

Dataset	Backbone	Eligible trajectories	Eligible steps
HotpotQA	Qwen3-4B	246/500 (49.2%)	984/1,725 (57.0%)
HotpotQA	Llama-3.2-3B	328/500 (65.6%)	1,312/1,888 (69.5%)
2Wiki	Qwen3-4B	203/500 (40.6%)	812/1,749 (46.4%)
2Wiki	Llama-3.2-3B	293/500 (58.6%)	1,172/1,917 (61.1%)
MuSiQue	Qwen3-4B	377/500 (75.4%)	2,607/3,332 (78.2%)
MuSiQue	Llama-3.2-3B	366/500 (73.2%)	2,559/3,492 (73.3%)
Pooled		—	9,446/14,103 (67.0%)

Table 9: RQ3 sampled steps and valid judge responses used in Table 3. Bind and Spec are evaluated separately.

Dataset	Backbone	Sampled	Valid Bind	Valid Spec
HotpotQA	Qwen3-4B	500	500	500
HotpotQA	Llama-3.2-3B	500	500	500
2Wiki	Qwen3-4B	600	600	600
2Wiki	Llama-3.2-3B	500	252	253
MuSiQue	Qwen3-4B	500	248	247
MuSiQue	Llama-3.2-3B	500	254	253

fewer same-trajectory alternatives, which could make action recovery easier, but this effect is not measured here. Consequently, Bind values are not directly comparable across datasets with different horizon distributions. An alternative protocol could draw distractors across examples, but we retain same-trajectory distractors to keep the candidates within the same behavioral context.

When Binding Succeeds and When It Fails. Because the judge is shown only the rubric and the four candidates—the question and trajectory prefix are withheld—binding can succeed only when the rubric itself contains information that separates the gold action from its distractors. Figures 6 and 7 contrast the two outcomes, and both highlight the role of contrastive detail.

Figure 6 shows two successful cases. The first is a comparison question over two symmetric entities, where two candidates query the *other* entity instead. The rubric nonetheless states "Targets Will Durant's profession directly" and "Retrieves evidence that Durant was a philosopher," naming the relevant entity twice, while its third criterion ("Preserves budget for the second person") further signals that another entity remains to be handled. The second case is a terminal step whose rubric couples answer support with the forced-finish condition, identifying it as a submission rather than another retrieval. In both cases, the rubric contains enough action-specific information for the gold action to be recovered without the question.

Figure 7 shows the mirror image. These failures are informative: the rubrics are not nonsensical but describe plausible evaluation dimensions while omitting the contrastive details needed to identify the exact action. In the search-step case, the rubric asks for "the missing person's birth date," which correctly describes the *role* of the next search after one astronaut's birth date has been observed, but unlike the Will Durant case above, it never names Hans Schlegel as the missing entity. A symmetric distractor that searches for Bertalan Farkas's birth date is therefore also compatible with the criteria, leaving the gold action underdetermined by the rubric alone. In the finish-step case, the rubric asks the judge to submit a supported answer while avoiding guesses when evidence is weak. This makes the criteria conservative and less discriminative of the current action type, so a safer-looking search for the release date is selected instead of the gold `Finish` action.

These cases suggest a concrete direction for improving rubric generation: step-level rubrics could be made more action-discriminative by capturing features that distinguish the intended action from

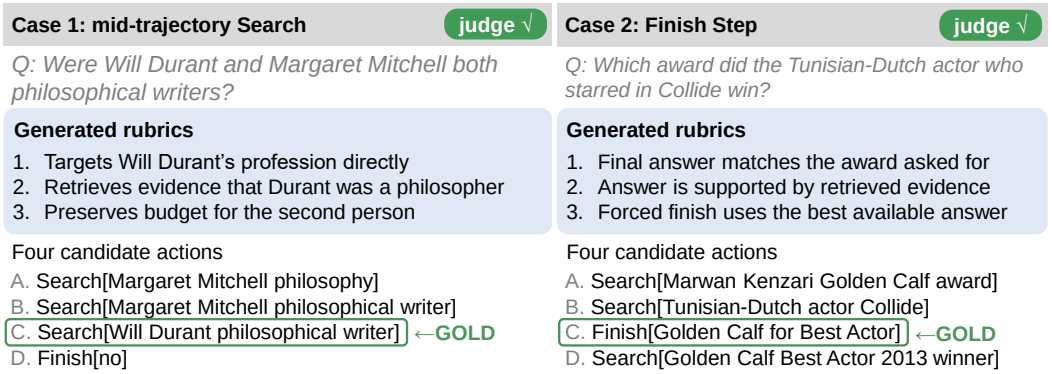


Figure 6: Two successful binding cases from HotpotQA validation (Qwen3-4B, ARCO epoch 4): a mid-trajectory `Search` and a terminal `Finish`. The question is shown here only for the reader—the judge sees *neither* the question nor the trajectory prefix, and must pick the gold action (GOLD) from the rubric and four candidates alone. In both cases the rubric contains content specific to the gold action, allowing the judge to identify it despite closely related distractors.

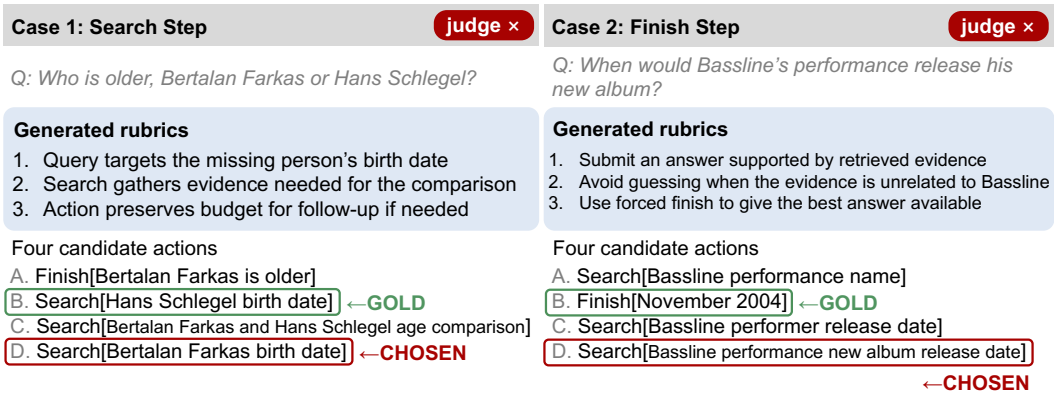


Figure 7: Two RQ3 rubric-binding failures. In Case 1, the gold action is `Search[Hans Schlegel birth date]`, but the rubric's phrase "missing person's birth date" does not explicitly name Hans Schlegel, so the judge selects the symmetric distractor `Search[Bertalan Farkas birth date]`. In Case 2, the gold action is a forced `Finish[November 2004]`, but the rubric emphasizes avoiding unsupported guesses, making an additional release-date search appear more aligned with the criteria.

nearby alternatives. For search steps, criteria could explicitly mention the target entity, relation, or missing evidence item that separates the current query from previous or alternative searches; for finish steps, they could more clearly encode answer readiness and action type, stating that the step is judged as a final submission rather than another evidence-gathering action. Contrastive rubric training with same-trajectory negatives, or prompting the rubric generator to name the distinguishing entity or relation, may therefore improve step binding.

D.2 Rubric Redundancy (RQ4)

Figure 8 shows one validation step for each of the three redundancy patterns discussed in §4.5, illustrating how criteria overlap as rubric width K grows.

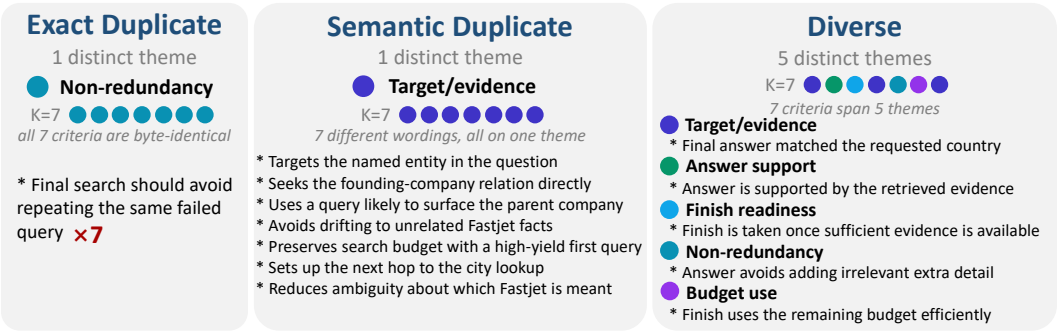


Figure 8: Three $K=7$ HotpotQA / Qwen validation steps with different redundancy patterns. Colored dots indicate the semantic theme assigned to each criterion. *Exact duplicate*: all 7 criteria are byte-identical. *Semantic duplicate*: 7 different wordings are clustered into a single theme. *Diverse*: 7 criteria span 5 distinct themes.

E PROMPTS

This appendix lists the prompts used by ARCO. The three multi-hop QA datasets share the same prompt structure and differ only in the dataset task description and a few constants such as the search budget, so we show the HotpotQA version for the policy and rubric prompts. Braces in the templates below are Python format placeholders; doubled braces (e.g., `{{"criteria": ...}}`) escape literal braces, so the string received by the model contains single braces. The width placeholders expand from K: with the default K=3, `{K_schema}` becomes `"<criterion 1>", "<criterion 2>", "<criterion 3>"` and `{K_score_schema}` becomes `<float>, <float>, <float>`. The step-level rubric prompt is used during RL and evaluation, while the trajectory-level rubric prompt is used only for warmup annotation: it asks the teacher to write per-step criteria and decomposition scores, which are subsequently projected to satisfy $\sum_t s_t = R(\tau)$ exactly (Appendix A). Finally, we include the two judge prompts for the RQ3 rubric-quality diagnostics (§4.4): the four-way step-binding judge (**Bind**) and the step-specificity judge (**Spec**).

Policy Prompt (HotpotQA)

[SYSTEM]
You are a multi−hop question answering agent. Your goal is to answer complex questions that require gathering information from multiple sources.

Available actions:
− Search[query] : Search for information using a short query.
− Finish[answer] : Submit your final answer.

Guidelines:
− Keep the entire output under 64 tokens.
− You have a limited search budget. Use it to gather the facts you need.
− Do not repeat a search you have already performed.
− Use Finish[answer] as soon as you are confident in the answer.
− For Search[query], use a short query phrase, not a full sentence.
− For yes/no questions, use Finish[yes] or Finish[no].
− For other questions, be concise but complete (e.g. "Owings Mills, Maryland" not just "Maryland").
− Output exactly one action. Do not output multiple actions.
− Output exactly one line in one of these formats, with no angle brackets:
Search[query]
Finish[answer]

[USER]
Question: {question}

Previous steps:
{history}

You have {searches_remaining} search(es) remaining. Use them wisely before submitting your answer.

[FORCED FINISH USER]
Question: {question}

Previous steps:
{history}

You have no searches remaining. You must now submit your final answer.
Use Finish[answer] −− do not search again.

Figure 9: Policy prompt for the HotpotQA agent. The forced-finish variant is used when the search budget is exhausted.

Rubric System Prompt

```
You are an expert evaluator for an agent acting in an interactive task.
Your job is to identify what an action should be judged on in context.

{task_context}

Generate local, context−specific evaluation criteria for the action being evaluated.
These criteria should define what the action ought to be judged on in context.
They may be dimensions, rules, or principles, depending on the task.

The criteria are not a rationale for the score. Do not explain why the model deserves a score. Instead,
output the evaluation dimensions, rules, or principles themselves.
Output only the final JSON requested by the user prompt, with no extra text.
```

Figure 10: Shared rubric system prompt defining the evaluator role and output constraints.

Step-Level Rubric Prompt

```
Question: {question}

Trajectory so far:
{history}

Searches remaining before this action: {searches_remaining}

Action being evaluated: {action}

Generate exactly {K} local criteria for judging this action. The criteria should say what this step
ought to be judged on, such as query targeting, planning quality, evidence gathering, answer
readiness, or avoiding redundant searches.
If the action is malformed or marked as invalid, judge it as an invalid−action step that wastes budget
and should be corrected on the next turn.
If searches remaining is 0, the agent should submit with Finish[answer]; a Search action should be
judged as a search−budget violation.
Each criterion should be short and concrete. Keep the full output concise.
Write criterion phrases, not explanations.

Output valid JSON only:
{{"criteria": [{K_schema}]}}
```

Figure 11: Step-level rubric prompt generating local criteria for an individual policy action.

Warmup Trajectory Rubric Prompt

Below is a complete trajectory and the final reward it received.
Question: {question}
Trajectory:
{trajectory_text}
Final reward: {reward:.3f} (0.0 = completely wrong, 1.0 = fully correct)

For each step, write exactly {K} evaluation criteria for that action based on the context available at that point. The criteria should say what this step ought to be judged on, such as query targeting, planning quality, evidence gathering, answer readiness, or avoiding redundant searches. Then assign a score in [−1, +1] for each criterion separately.
If a step is marked as an invalid action or its observation reports an invalid action, judge it as a malformed non−executable step that wastes budget.
If a step is marked as a forced−finish violation, judge it as an invalid final action caused by searching after the search budget was exhausted.
The criteria should be local to the current prefix. Different steps should usually have different criteria.

Do not write a rationale for the score. Write the criteria themselves.
Each criterion should be short and concrete. Keep the full output concise.
Write criterion phrases, not explanations.
Output only the final JSON array.

Scoring guide (range −1 to +1, applies to each individual criterion score):
− Scores are decomposition credits, not standalone quality labels. The final reward is only {reward:.3 f}, so do not give large positive scores to many steps unless there are negative scores elsewhere to compensate.
− +0.7 to +1.0 : Very strong positive credit −− use sparingly for a criterion that is essential to the final correct answer.
− +0.2 to +0.6 : Helpful −− this criterion moves the trajectory toward the answer.
− 0.0 : Neutral −− this criterion neither helps nor hurts.
− −0.2 to −0.6 : Harmful or wasteful −− this criterion wastes budget, misses evidence, or adds confusion.
− −0.7 to −1.0 : Strongly harmful −− this criterion causes a wrong final answer or violates the search budget.

CONSTRAINT: Each step must have exactly {K} criteria and exactly {K} scores. The average of the {K} scores is the step's overall score. The sum of all step averages must equal exactly {reward:.3f}. This hard decomposition constraint overrides the local scoring guide: adjust magnitudes, including negative scores when necessary, so the exact sum is {reward:.3f}.
CONSTRAINT: Output exactly one JSON entry for each step in the provided trajectory, in the same order. Do not add extra steps, summaries, or overall entries. Do not omit any step.

Output a JSON array −− one entry per step, in order. Even if there is only one step, wrap it in an array. No markdown fences. No extra text.

```
[
  {{"step": 0, "criteria": [{K_schema}], "scores": [{K_score_schema}]}},
  {{"step": 1, "criteria": [{K_schema}], "scores": [{K_score_schema}]}},
  ...
]
```

Figure 12: Trajectory-level rubric prompt used during warmup annotation to generate criteria and decomposition-compatible pseudo scores.

RQ3 Step-Binding Judge Prompt

You are evaluating whether a generated rubric is action−specific.

You are given only the generated rubric and four candidate actions. The question and trajectory context are intentionally hidden.

Your task is to select which candidate action this rubric was most likely generated to evaluate. This condition measures whether the rubric itself points to a concrete action rather than being generic.

Important rules:
− Use only the rubric and candidate actions.
− Do not reward a generally good action unless the rubric specifically matches it.
− Use the content, intent, and stage of each action to determine which one the rubric most directly evaluates.

Generated rubric:
{rubric_bullets}

Candidate actions:
{candidate_actions}

Output JSON only, with no explanation:
{{"choice": "<A|B|C|D>"}}

Figure 13: RQ3 four-way step-binding judge prompt (**Bind**). The judge sees only the rubric and four candidate actions; the question and trajectory prefix are hidden, so success requires the rubric itself to identify the gold action.

RQ3 Step-Specificity Judge Prompt

You are judging whether a generated rubric provides local, context−specific criteria for evaluating an agent step.

Question:
{question}

Trajectory before the current step:
{history}

Searches remaining before this action: {searches_remaining_before}
Step index: {step_idx}

Action being evaluated:
{gold_action}

Generated rubric:
{rubric_bullets}

Rate the rubric from 1 to 5:
1 = generic or unrelated to the action
2 = weakly related but mostly generic
3 = partially action−relevant
4 = clearly action−specific and context−aware
5 = highly specific, context−grounded, and directly useful for scoring this step

Output JSON only, with no explanation:
{{"fit": <1−5>}}

Figure 14: RQ3 step-specificity judge prompt (**Spec**). The judge sees the question, trajectory prefix, gold action, and rubric, and rates how specifically the rubric evaluates that action on a 1–5 scale.