RACA: ROLE-AWARE CREDIT ASSIGNMENT FOR MULTI-AGENT LLM REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent research has sought to enhance LLM reasoning through collaboration among specialized agents. Training such systems, however, requires accurate credit assignment across roles and decisions. Existing outcome-level reinforcement learning assigns the same terminal reward to every turn, allowing an incorrect action to receive positive credit when later agents repair it. We introduce Role-Aware Credit Assignment (RACA), which aligns credit with each role's responsibility and decision scale. RACA retains trajectory-level credit for controller decisions, evaluates worker actions using locally observable outcomes, and computes relative advantages only among turns matched by role, blackboard-derived context, and reward channel. For primary proposer responses, it separately routes solution-quality and assistance-seeking advantages to their corresponding token spans. All signals come from the original rollout, without an additional value model or counterfactual execution. Using Qwen3-8B, RACA achieves 64.5% accuracy on MATH Level 5 and 16.0% on AIME 2022–2026, outperforming all compared baselines on both benchmarks. Under the same adaptive multi-agent framework, it improves over Outcome GRPO by 6.4 and 6.0 percentage points, respectively. On MATH, RACA averages 7.4 model calls per problem, compared with 8.7–10.7 for the other adaptive methods.

1 INTRODUCTION

Large language models (LLMs) have made substantial progress in complex reasoning by generating intermediate steps before a final answer (Wei et al., 2022; Guo et al., 2025). Building on these advances, multi-agent systems organize model instances with complementary responsibilities as collaborative problem solvers (Motwani et al., 2024; Wan et al., 2026). Through debate, critique, and verification, they can identify and correct errors that a single model may miss (Du et al., 2023; Liang et al., 2024), offering a flexible way to combine reasoning diversity with problem-dependent computation.

However, many prompting-based systems rely on manually specified protocols, such as repeated debate, fixed role assignments, and predefined aggregation rules (Du et al., 2023; Liang et al., 2024; Chan et al., 2024). These fixed interaction patterns may introduce redundant review when an answer is reliable yet provide insufficient assistance when errors remain unresolved. For tasks with automatically verifiable answers, reinforcement learning can instead learn adaptive collaboration from task outcomes (Wan et al., 2026; Park et al., 2025; Zhao et al., 2026). Extending this approach to specialized agents, however, introduces a fundamental credit-assignment problem (Foerster et al., 2018; Li et al., 2026).

Outcome-level optimization normalizes terminal correctness rewards across sampled trajectories and assigns the resulting advantage to every decision within each trajectory (Shao et al., 2024; Guo et al., 2025). In heterogeneous multi-agent systems, this obscures individual contributions: an incorrect proposal may receive positive credit after subsequent collaboration repairs it, while useful criticism may receive negative credit when another role causes the episode to fail, as illustrated in Fig. 1(b). Agent- or turn-level methods provide finer granularity (Feng et al., 2026a; Zhao et al., 2026; Feng et al., 2026b), but typically retain one scalar advantage per turn and may compare actions from different reasoning states. A primary proposer response exposes a further limitation: it both generates a solution and decides whether to request assistance. A correct answer may accompany

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.

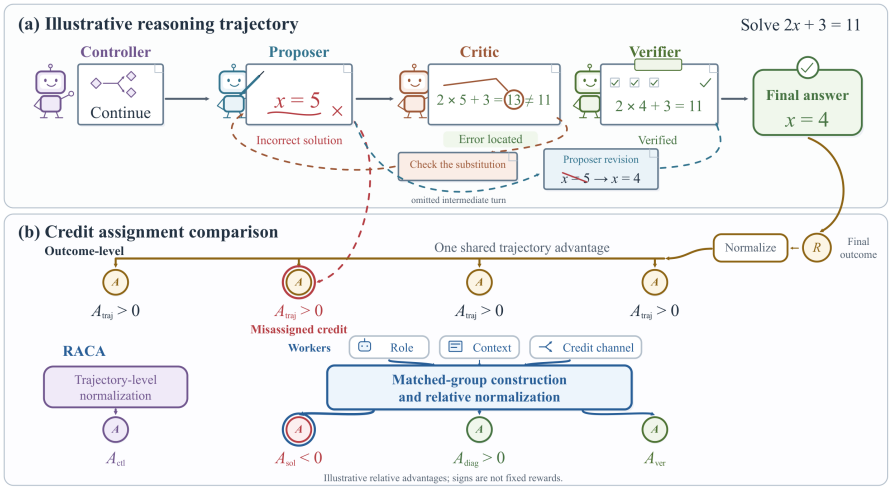


Figure 1: Motivating example of credit misassignment in multi-agent reasoning. (a) The controller, proposer, critic, and verifier collaborate to detect, revise, and verify an initially incorrect solution. Dashed arrows denote intermediate interaction turns omitted for clarity. (b) Outcome-level optimization broadcasts the same trajectory advantage to every turn, whereas RACA retains trajectory-level normalization for the controller and normalizes each worker turn only against turns with the same role, reasoning context, and reward channel. Advantage signs are illustrative rather than fixed rewards.

an unnecessary request, while an incorrect answer may appropriately seek help. These decisions therefore require distinct credit signals even within the same turn.

More precise credit could be obtained through PPO with a learned value function (Schulman et al., 2017) or a process reward model (Lightman et al., 2024; Wang et al., 2024), but these approaches require either estimating returns across heterogeneous role states or costly supervision with role-specific targets. We instead introduce *Role-Aware Credit Assignment* (RACA), which derives credit directly from observed rollout outcomes. RACA aligns rewards and comparison groups with each role's responsibility while separating function-specific credit within primary proposer responses. Our main contributions are:

(1) We distinguish episode-level control from local worker actions. RACA retains trajectory-level normalization for the controller and assigns workers local rewards according to their responsibilities.

(2) We form worker comparison groups matched by role, blackboard context, and reward channel. Within primary proposer responses, solution and interaction rewards are normalized independently, and their advantages are routed to disjoint token spans. All signals come from the original rollout, without a learned value function, a process reward model, or counterfactual execution.

(3) On MATH Level 5 and AIME 2022–2026, RACA outperforms all compared baselines, improving over Outcome GRPO by 6.4 and 6.0 percentage points under the same adaptive framework. On MATH, it also uses fewer model calls than the other adaptive methods.

2 Related Work

Reinforcement Learning for LLM Reasoning. Reinforcement learning advances LLM reasoning through verifiable task outcomes. DeepSeekMath introduces Group Relative Policy Optimization (GRPO), which estimates advantages within sampled response groups without a learned value model, while DeepSeek-R1 demonstrates the effectiveness of large-scale reasoning RL (Shao et al., 2024; Guo et al., 2025). Beyond final-answer rewards, process supervision evaluates intermediate steps (Lightman et al., 2024), VinePPO uses Monte Carlo value estimates from partial-trajectory continuations for step-level credit (Kazemnejad et al., 2024), and SCoRe encourages multi-turn self-correction through reward shaping (Kumar et al., 2025). These methods refine credit across so-

lutions, steps, and revisions within a single reasoning policy. Role-specialized systems additionally require credit to reflect each decision's function.

Multi-Agent LLM Reasoning and Learning. Prompted debate, iterative discussion, and role-based evaluation enable agents to exchange solutions and critiques without training their role-specific behaviors (Du et al., 2023; Liang et al., 2024; Chan et al., 2024). Recent work moves toward learned collaboration: MALT propagates downstream outcomes through generator–verifier–refiner trees to construct role-specific post-training data (Motwani et al., 2024), and ReMA learns hierarchical meta-thinking and reasoning through multi-agent RL (Wan et al., 2026). MAPoRL uses verifier-based rewards for corrective revisions and beneficial influence on other agents (Park et al., 2025), while Dr. MAS stabilizes training with agent-wise advantage normalization (Feng et al., 2026b). These approaches advance specialization and coordination, but collaborative success does not by itself resolve how credit should be distributed across role-specific actions or distinct functions within one response.

Credit Assignment for LLM Agents. Credit assignment identifies useful decisions under sparse, delayed rewards. COMA estimates agent contributions using a centralized critic and a counterfactual baseline that holds other agents' actions fixed (Foerster et al., 2018); SHARP uses Shapley-inspired counterfactual masking for multi-agent LLM systems (Li et al., 2026). Such approaches separate individual contributions from shared outcomes but require a learned critic or additional counterfactual evaluations. Critic-free alternatives refine group-relative advantages: GiGPO combines episode-level credit with step-level comparisons among recurring environment states (Feng et al., 2026a), while AT-GRPO groups by agent identity and turn position and combines team outcomes with local incentives (Zhao et al., 2026). However, finer agent- or step-level credit does not by itself ensure comparisons between functionally equivalent actions. A single scalar advantage for an entire response also leaves solution generation and assistance-seeking coupled, even when these decisions warrant different credit signals.

3 METHOD

We introduce *Role-Aware Credit Assignment* (RACA), which aligns credit with the responsibilities of specialized agents. As illustrated in Figure 2, RACA evaluates controller decisions at the episode level and worker actions through local outcomes, then computes relative advantages within role-, context-, and channel-matched comparison groups. For primary proposer responses, solution generation and assistance-seeking receive separate advantages applied to their corresponding token spans. We first describe the multi-agent setup, then present the role-specific rewards and credit-assignment mechanism.

3.1 MULTI-AGENT SETUP

We consider reasoning tasks with automatically checkable answers. Given a question x with reference answer $a^\star$, four role-specialized agents collaborate for at most T active reasoning rounds: a controller π_{ctl}, a proposer π_{pro}, a critic π_{cri}, and a verifier π_{ver}. The controller decides whether to continue reasoning, the proposer generates or revises solutions, the critic identifies errors, and the verifier estimates answer correctness. The agents share a frozen language-model backbone with separate trainable low-rank adapters, activating only the corresponding role's adapter for each call.

The collaboration forms a trajectory τ, with each role call constituting a turn. Agents communicate through a shared blackboard B_t that records candidate solutions, criticism, verification scores, and interaction events. From B_t we deterministically derive a per-round semantic context $\xi_t \in \{\texttt{explore}, \texttt{refine}, \texttt{verify}\}$ from the append order of the blackboard: `explore` when no candidate solution has been written, `refine` when the most recent write is a criticism rather than a solution, and `verify` otherwise (App. B). This context conditions the comparison groups in Sec. 3.3. Each round begins with a controller decision. If reasoning continues, the proposer produces a candidate answer a_t and an interaction decision $u_t \in \{\texttt{none}, \texttt{request}\}$ in a primary turn, selecting the critic or verifier when requesting assistance. The solution precedes the interaction block in the generated response, so assistance-seeking is conditioned on the solution prefix.

A request initiates a chain of response turns that update the blackboard and may request further interaction, subject to a per-round budget of H hops. Detected errors trigger proposer corrections

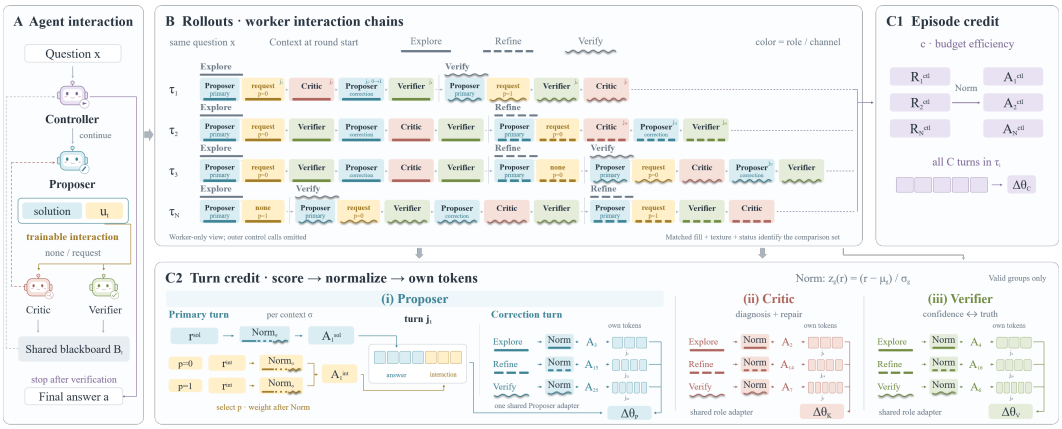


Figure 2: Overview of RACA. (A) Specialized agents collaborate through a shared blackboard. (B) Worker interaction chains from N trajectories sampled for the same question, with contexts determined at round start. (C1) Controller turns receive episode-level advantages. (C2) Worker rewards are normalized within role-, context-, and channel-matched groups. Primary proposer solution and interaction advantages are applied to their corresponding token spans.

when budget remains. An accepted controller stop ends the episode before another primary turn; otherwise, reasoning continues until the round limit. The trajectory factorization and execution rules, including the verification requirement for stopping, are provided in Appendices A and B.

At termination, a uniform majority vote over accumulated candidate answers determines the final answer a. We distinguish episode correctness, $c = \mathrm{match}(a, a^\star)$, from the correctness of each primary answer before interaction, $p_t = \mathrm{match}(a_t, a^\star)$, where match is the deterministic binary answer matcher used in training and evaluation. This distinction allows local proposal quality to be evaluated separately from the outcome of subsequent collaboration.

During training only, interactions are also injected into a subset of rounds in which the proposer declines assistance. These rounds retain solution credit and train responding roles normally, but are excluded from both interaction-channel normalization and interaction-decision updates. Writing forced_t for the injection indicator, we express this eligibility as $m_t^{\mathrm{int}} = 1 - \mathrm{forced}_t$. The injection schedule and target-selection rules are detailed in Appendix B.

3.2 Role-Specific Turn Rewards

The process in Sec. 3.1 produces decisions with different responsibilities and temporal scopes. Rather than assigning a single terminal reward to all turns, RACA evaluates worker turns through role-specific local outcomes and the controller through episode correctness and efficiency.

For a primary proposer turn, solution quality and assistance-seeking are evaluated separately. The solution reward measures answer correctness before interaction, $r_t^{\mathrm{sol}} = p_t$. To evaluate the interaction decision u_t, let $\widetilde{p}_t$ denote the correctness of the candidate remaining at the end of round t, after any correction through the interaction chain. For proposer-chosen interaction decisions, we define

$$r_t^{\mathrm{int}} = \begin{cases} g - c_{\mathrm{int}}, & u_t \neq \texttt{none},\ p_t = 0,\ \widetilde{p}_t = 1, \\ -c_{\mathrm{int}}, & u_t \neq \texttt{none},\ p_t = 0,\ \widetilde{p}_t = 0, \\ -c_{\mathrm{int}} - o, & u_t \neq \texttt{none},\ p_t = 1, \\ 0, & u_t = \texttt{none},\ p_t = 1, \\ -m, & u_t = \texttt{none},\ p_t = 0. \end{cases} \tag{1}$$

Here $g > 0$ rewards successful repair, $c_{\mathrm{int}} \geq 0$ is the communication cost, and $o, m > 0$ penalize unnecessary requests after correct answers and missed requests after incorrect answers, respectively. The solution and interaction rewards remain separate: each is normalized within its own comparison group and applied to its corresponding token span, as detailed in Sec. 3.3.

Critic-triggered proposer corrections receive only revised-answer correctness rewards. They may request further interaction within the remaining hop budget, but receive no separate interaction reward. The correction reward is formalized in Appendix C.

The critic is evaluated on error detection and subsequent repair. Let $f_{t,h} \in \{0,1\}$ indicate whether it flags an error, $p^{\text{rev}}_{t,h} \in \{0,1\}$ the reviewed answer's correctness, and $q_{t,h} \in \{0,1\}$ the short-horizon correction outcome following a correctly flagged error. The reward is

$$r^{\text{cri}}_{t,h} = \begin{cases} b_{\text{tp}} + g_{\text{cri}} q_{t,h}, & f_{t,h}=1,\ p^{\text{rev}}_{t,h}=0,\ q_{t,h} \text{ available}, \\ b_{\text{terminal}}, & f_{t,h}=1,\ p^{\text{rev}}_{t,h}=0,\ q_{t,h} \text{ unavailable}, \\ -\eta_{\text{fp}}, & f_{t,h}=1,\ p^{\text{rev}}_{t,h}=1, \\ b_{\text{tn}}, & f_{t,h}=0,\ p^{\text{rev}}_{t,h}=1, \\ 0, & f_{t,h}=0,\ p^{\text{rev}}_{t,h}=0. \end{cases} \tag{2}$$

Here b_{tp} rewards correct detection, g_{cri} rewards observed repair, η_{fp} penalizes false alarms, and b_{tn} rewards correctly withholding a flag. When no correction outcome is available, the fixed reward b_{terminal} replaces the repair-dependent reward rather than using final episode correctness. The repair signal reflects observed local outcomes, not a counterfactual causal effect. The outcome window and its availability conditions are specified in Appendix C.

While the critic diagnoses errors, the verifier estimates answer correctness through a confidence score $v_{t,h} \in [0,1]$. Its reward is one minus the Brier score, $r^{\text{ver}}_{t,h} = 1 - (v_{t,h} - p^{\text{rev}}_{t,h})^2$, where $p^{\text{rev}}_{t,h}$ now refers to the answer reviewed by the verifier. This reward is maximized in expectation by reporting the conditional probability of correctness; an unparseable score receives zero reward. The score enters the blackboard as evidence, whereas the reward is used only to update the verifier policy.

The controller, in contrast, is evaluated at the episode level. Let R be the number of active proposer rounds and $\varrho = (T-R)/T$ the unused round fraction. We define

$$r^{\text{ctl}} = c + \alpha c \varrho - \beta(1-c) - \gamma(1-c)\varrho. \tag{3}$$

Correct episodes receive a bonus for unused budget, whereas incorrect episodes incur a penalty that increases with the unused fraction. Thus, efficiency is rewarded only together with correctness, rather than encouraging early stopping independently of the outcome. The resulting episode-level advantage is assigned to every sampled controller proposal, while worker rewards remain turn-specific. Reward attachment rules are given in Appendix C.

3.3 Advantage Estimation and Optimization

The rewards in Sec. 3.2 evaluate different functions and temporal scopes, so they cannot share a single normalization baseline. RACA instead computes group-relative advantages within role-conditioned semantic groups. All comparisons use the N trajectories sampled for the same question; turns from different questions are never mixed.

For the controller, the comparison unit is the complete episode. Every controller turn in trajectory i receives

$$A^{\text{ctl}}_i = \frac{r^{\text{ctl}}_i - \mu_{\text{ctl}}}{\sigma_{\text{ctl}}}, \tag{4}$$

where r^{ctl}_i is defined by Eq. (3), and μ_{ctl} and σ_{ctl} are computed across the N trajectories. The controller group is omitted when σ_{ctl} falls below the threshold δ.

Worker turns require finer comparisons. We characterize each turn by its role, blackboard-derived context, and response status through $b(j) = (k(j), \xi_{t(j)}, \iota_j)$, where $\iota_j = 0$ denotes a primary turn and $\iota_j = 1$ denotes a response turn, including auxiliary verifier turns. Critic, verifier, and proposer correction responses each contribute a single reward channel with anchor $g_{\text{rew}}(j) = (b(j), \text{rew})$. This separates rewards across roles and reasoning contexts, while distinguishing proposer corrections from primary solution attempts.

For a primary proposer turn, the two functions require different comparison sets. The solution anchor is $g_{\text{sol}}(j) = (\text{pro}, \xi_{t(j)}, 0, \text{sol})$. It deliberately excludes p_t: grouping by correctness would

make the binary solution reward constant within each group and eliminate the signal distinguishing correct from incorrect answers. The interaction anchor, in contrast, conditions on primary-answer correctness:

$$g_{\text{int}}(j) = \big(\text{pro}, \xi_{t(j)}, 0, \text{int}, p_{t(j)}\big). \tag{5}$$

Because the solution prefix precedes the interaction decision, p_t characterizes the context before that decision rather than its outcome. Conditioning on p_t separates unnecessary requests after correct answers from potentially useful requests after incorrect answers, so interaction choices are compared at the same level of solution quality.

For channel c, let $\mathcal{G}_{g_c}$ contain turns matching anchor g_c. The channel-specific advantage is

$$A^i_{j,c} = \frac{r^i_{j,c} - \mu_{g_c}}{\sigma_{g_c}}, \tag{6}$$

where μ_{g_c} and σ_{g_c} are computed after within-round deduplication of repeated same-role outcomes. This prevents repeated calls from receiving extra weight in the group statistics; the representative-selection rules are given in Appendix D. A worker channel is retained only if its group has at least two representatives and $\sigma_{g_c} > \delta$. Forced primary turns are excluded from $\mathcal{G}_{g_{\text{int}}}$ but remain eligible for the solution channel.

The resulting proposer advantages remain separate when applied to tokens. Let $\mathcal{S}^i_{j,\text{sol}}$ and $\mathcal{S}^i_{j,\text{int}}$ denote the disjoint spans of solution text and the trailing interaction block. Their weighted advantages are

$$\overline{A}^i_{j,c} = \begin{cases} A^i_{j,\text{sol}}, & c = \text{sol}, \\ \lambda_{\text{int}} A^i_{j,\text{int}}, & c = \text{int}, \end{cases} \qquad \mathcal{S}^i_{j,\text{sol}} \cap \mathcal{S}^i_{j,\text{int}} = \varnothing. \tag{7}$$

Let $\mathcal{C}^i_j$ contain the retained channels of turn j. The primary solution channel is included whenever its group is valid, whereas the interaction channel additionally requires $m^{\text{int}}_{t(j)} = 1$. For controller and other worker turns, the retained channel spans the full response, with $\overline{A}^i_{j,c} = A^{\text{ctl}}_i$ for controller turns and $\overline{A}^i_{j,c} = A^i_{j,\text{rew}}$ for worker turns.

We optimize these channel-routed advantages with a clipped policy objective. Let $\mathcal{T}_i$ contain turns with at least one valid advantage, including eligible auxiliary verifier turns, and let $M = \sum_i |\mathcal{T}_i|$. Writing $\rho^i_{j,l}$ for the token-level probability ratio of the current role policy to its rollout policy, RACA maximizes

$$\mathcal{J}(\theta) = \frac{1}{M} \sum_i \sum_{j \in \mathcal{T}_i} \left[\sum_{c \in \mathcal{C}^i_j} \frac{1}{|\mathcal{S}^i_{j,c}|} \sum_{l \in \mathcal{S}^i_{j,c}} \min\Big(\rho^i_{j,l} \overline{A}^i_{j,c}, \text{clip}\big(\rho^i_{j,l}, 1-\epsilon, 1+\epsilon\big) \overline{A}^i_{j,c} \Big) - \frac{\kappa}{L^i_j} \sum_{l=1}^{L^i_j} \widehat{D}^i_{j,l} \right]. \tag{8}$$

Each channel is averaged over its own token span before the contributions are combined. Thus, the short interaction block is not diluted by the longer solution text, and neither advantage is applied outside its corresponding span. Single-channel turns reduce to the usual turn-normalized update. Degenerate channels are omitted rather than assigned zero advantages, and turns with no valid channel are excluded from M.

Here ϵ is the clipping parameter and $\widehat{D}^i_{j,l}$ is the nonnegative per-token KL estimator against the role's frozen SFT reference policy, weighted by κ. Each turn updates only its role adapter; the backbone and reference adapters remain frozen. The probability ratio, KL estimator, and reference-policy construction are detailed in Appendix D.

4 Experiments

We evaluate RACA on MATH and AIME using Qwen3-8B, examining reasoning accuracy, interaction selectivity, inference efficiency, and the contributions of its credit-assignment components.

4.1 Experimental Settings

Benchmarks. We use 5,265 MATH (Hendrycks et al., 2021) Levels 3–5 problems for disjoint training and validation partitions. Evaluation covers 1,000 MATH Level-5 test problems and all

Table 1: Final-answer accuracy on held-out MATH Level-5 and AIME sets (test sizes in parentheses). Adaptive multi-agent methods differ only in credit assignment. RL results report mean ± standard deviation over three independent training runs. Best and second-best results are **bold** and underlined, respectively.

Method	Optimization	Inference Strategy	MATH-L5 (1000) ↑	AIME 2022–2026 (150) ↑
Single-agent baselines				
SFT-CoT	SFT only	Greedy single trajectory	46.2	6.7
Single-Agent GRPO	Outcome GRPO	Greedy single trajectory	54.9 ± 0.7	9.3 ± 0.6
Test-time reasoning baselines				
Self-Consistency	SFT only	$N = 8$ majority vote	57.5	8.7
Single-Agent GRPO + Self-Consistency	Outcome GRPO	$N = 8$ majority vote	$\underline{62.4 \pm 0.3}$	14.4 ± 0.6
Self-Refine	SFT only	One critique–revision cycle	50.2	8.7
Fixed multi-agent baselines				
Fixed Four-Role Pipeline	SFT only	Fixed role sequence	48.6	6.7
Fixed Four-Role + GRPO	Outcome GRPO	Fixed role sequence	53.8 ± 0.9	12.7 ± 1.3
Adaptive multi-agent credit assignment				
Multi-Agent Outcome GRPO	Trajectory	Adaptive blackboard	58.1 ± 0.8	10.0 ± 0.7
Role-Wise Reward GRPO	Role	Adaptive blackboard	60.0 ± 0.4	10.7 ± 1.1
AT-GRPO	Agent + turn	Adaptive blackboard	60.8 ± 0.6	12.0 ± 1.3
GiGPO	Episode + anchor state	Adaptive blackboard	59.1 ± 1.0	$\underline{14.7 \pm 0.7}$
RACA (Ours)	**Role + context + functional channel**	**Adaptive blackboard**	$\mathbf{64.5 \pm 0.6}$	$\mathbf{16.0 \pm 1.3}$

150 AIME I/II problems from 2022–2026, with AIME assessing generalization beyond MATH. No AIME problem is used for SFT or RL, and neither test set is used for model selection or tuning. All methods share the evaluation protocol, with final-answer accuracy as the primary metric. Data preparation and evaluation details appear in Appendix E.1.

Baselines. We compare three groups: single-agent methods (SFT-CoT (Wei et al., 2022) and Single-Agent GRPO (Shao et al., 2024)); test-time reasoning and fixed coordination (Self-Consistency (Wang et al., 2022) on both checkpoints, Self-Refine (Madaan et al., 2023), and SFT and outcome-GRPO variants of the Fixed Four-Role Pipeline); and credit-assignment methods within the same adaptive four-role blackboard framework (Multi-Agent Outcome GRPO, Role-Wise Reward GRPO, AT-GRPO (Zhao et al., 2026), and GiGPO (Feng et al., 2026a)). Implementation details are provided in Appendix E.3.

Models and Training Settings. All methods use Qwen3-8B (Yang et al., 2025). Each role-specific LoRA adapter (Hu et al., 2021) is separately supervised fine-tuned on demonstration turns corresponding to its role, with the shared backbone frozen. The resulting role-specific SFT checkpoints provide the same initialization for corresponding roles across methods. Multi-agent methods optimize all four role adapters during RL, with each turn updating only its corresponding adapter; the single-agent baseline optimizes only the proposer adapter.

Unless otherwise specified, RL methods share the data partition, sampling and optimizer settings, maximum training budget, stopping criterion, and evaluation protocol. Training proceeds until convergence, monitored exclusively on the MATH validation partition, which also determines checkpoint selection. Sampling, interaction budgets, and decoding settings are detailed in Appendix E.7.

4.2 Main Results

Question 1. Does RACA outperform single-agent and multi-agent reinforcement-learning baselines? Table 1 compares RACA with all baselines on the held-out MATH Level-5 and AIME test sets.

RACA achieves the highest accuracy on both benchmarks, reaching 64.5% on MATH Level 5 and 16.0% on AIME (Table 1). It outperforms the strongest MATH baseline, Single-Agent GRPO + Self-Consistency, by 2.1 percentage points and the strongest AIME baseline, GiGPO, by 1.3 points. Thus, its gains extend beyond comparisons with single-trajectory reasoning or fixed multi-agent coordination.

Under the same adaptive four-role blackboard framework, RACA improves over Multi-Agent Outcome GRPO by 6.4 and 6.0 percentage points on MATH and AIME, respectively. This comparison indicates that specialized roles and adaptive interaction alone do not fully account for the improvement, supporting the benefit of aligning credit with role responsibilities, reasoning contexts, and functional decisions. Additional baseline comparisons are discussed in Appendix F.

Table 2: Interaction behavior on MATH Level 5. Pre-interaction accuracy is computed from the initial proposer answer, while post-interaction accuracy is computed from the final system answer. Δ Acc. denotes the absolute accuracy change introduced by the subsequent multi-agent reasoning process. Interaction rates are measured at the reasoning-round level.

Method	**Pre-Int. Acc.**	**Post-Int. Acc.**	**Δ Acc. $\uparrow$**	**Int. Rate**	$P(\text{Int} \mid \text{Wrong}) \uparrow$	$P(\text{Int} \mid \text{Correct}) \downarrow$
Multi-Agent Outcome GRPO	57.1	58.1	+1.0	32.0	23.0	40.0
Role-Wise Reward GRPO	53.8	60.0	+6.2	65.0	39.0	86.0
AT-GRPO	55.1	60.8	+5.7	59.0	34.0	80.0
GiGPO	55.4	59.1	+3.7	42.0	31.0	53.0
RACA (Ours)	55.5	64.5	+9.0	61.0	58.0	63.0

Table 3: Accuracy–cost comparison on MATH Level 5. Language-model calls and token counts are averaged per problem and include all intermediate reasoning, feedback, and verification turns.

Method	**Accuracy $\uparrow$**	**LLM Calls $\downarrow$**	**Input Tokens $\downarrow$**	**Output Tokens $\downarrow$**
Single-agent and test-time reasoning baselines				
SFT-CoT	46.2	1.0	284.8	444.0
Single-Agent GRPO	54.9	1.0	284.8	419.9
Self-Consistency ($N = 8$)	57.5	8.0	2,278.4	3,456.5
Single-Agent GRPO + Self-Consistency ($N = 8$)	62.4	8.0	2,278.4	4,208.4
Self-Refine	50.2	3.0	1,793.6	1,027.4
Fixed multi-agent baselines				
Fixed Four-Role Pipeline	48.6	5.0	3,126.7	1,120.5
Fixed Four-Role + GRPO	53.8	5.0	2929.7	1,049.9
Adaptive multi-agent methods				
Multi-Agent Outcome GRPO	58.1	8.7	3878.3	1,606.7
Role-Wise Reward GRPO	60.0	10.7	4987.9	2,232.9
AT-GRPO	60.8	10.7	4928.8	2,193.1
GiGPO	59.1	9.3	4135.7	1,726.4
RACA (Ours)	64.5	7.4	3617.4	2014.3

4.3 Interaction and Efficiency Analysis

Question 2. Does RACA learn to interact selectively and efficiently? To distinguish selective collaboration from gains due to additional computation, we examine interaction behavior and inference cost on MATH Level 5. Table 2 reports accuracy before and after interaction and interaction rates conditioned on answer correctness. Table 3 compares accuracy, average model calls, and input and output token counts per problem.

RACA raises accuracy from 55.5% for the initial proposer answers to 64.5% for the final outputs, yielding the largest interaction gain of 9.0 percentage points (Table 2). Its interaction rate on incorrect answers is 58.0%, compared with 23.0–39.0% for the adaptive baselines. The gap between interaction rates on incorrect and correct answers narrows to -5.0 percentage points, versus -17.0 to -47.0 for the baselines. These results indicate improved relative selectivity, although RACA still interacts more frequently on correct answers (63.0%) than on incorrect ones.

This accuracy gain does not require more model calls. RACA averages 7.4 calls per problem, compared with 8.0 for both Self-Consistency variants and 8.7–10.7 for the adaptive multi-agent baselines. Relative to Single-Agent GRPO + Self-Consistency, it improves accuracy by 2.1 percentage points while using fewer calls and output tokens, although it processes more input tokens. Fixed pipelines require fewer calls but achieve substantially lower accuracy. RACA therefore offers a favorable accuracy–cost trade-off rather than uniformly minimizing every cost metric.

4.4 Credit Assignment and Learning Dynamics

Question 3. Does role-aware credit assignment provide more effective supervision than existing alternatives? Under the same adaptive framework and training protocol, RACA outperforms all credit-assignment baselines on both benchmarks (Table 1). We further examine its learning dynamics and individual components below.

Table 4: Ablation of the three principal components of RACA under the same adaptive four-role blackboard framework.

Variant	Role	Context	Channel	MATH-L5 ↑	AIME ↑
RACA w/o role conditioning	No	Yes	Yes	62.4 ± 0.4	15.3 ± 0.7
RACA w/o context conditioning	Yes	No	Yes	59.8 ± 1.0	13.3 ± 1.1
RACA w/o channel decomposition	Yes	Yes	No	60.6 ± 0.6	14.0 ± 1.3
RACA (Ours)	**Yes**	**Yes**	**Yes**	$\mathbf{64.5 \pm 0.6}$	$\mathbf{16.0 \pm 1.3}$

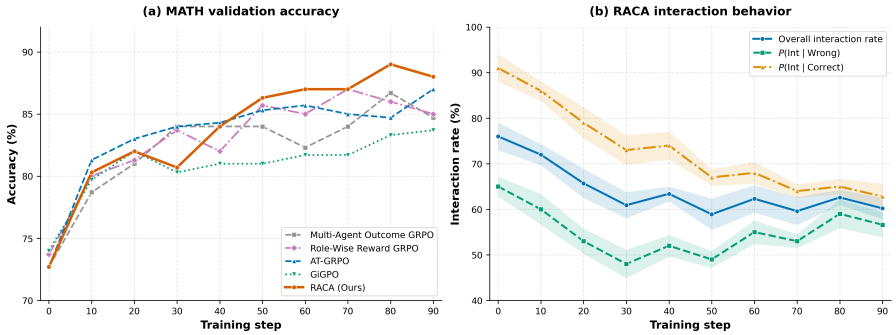


Figure 3: Learning dynamics on the MATH validation set. (a) Validation accuracy across training steps. (b) RACA's overall and correctness-conditioned interaction rates, measured per round for proposer-initiated requests only.

Figure 3(a) tracks MATH validation accuracy across training steps. RACA does not consistently lead early in training but achieves higher accuracy later.

Component ablations. We remove one component at a time while keeping the architecture, turn-level rewards, training data, and optimization protocol unchanged. Without role conditioning, rewards from different roles are normalized together. Without context conditioning, turns from different blackboard states share comparison groups. Without functional-channel decomposition, solution and interaction decisions in a primary proposer turn receive a single turn-level advantage.

Table 4 shows that all three components contribute to performance. Removing role conditioning reduces MATH and AIME accuracy by 2.1 and 0.7 percentage points, respectively. Removing functional-channel decomposition increases these drops to 3.9 and 2.0 points, supporting separate credit assignment for solution and interaction decisions. Context conditioning has the largest effect: removing it reduces accuracy by 4.7 and 2.7 points, highlighting the importance of comparing turns under matched reasoning contexts.

Interaction dynamics. Figure 3(b) shows that RACA's overall interaction rate declines during training. Requests on correct answers decrease substantially, while those on incorrect answers initially decline and then partially recover, narrowing the gap between the two conditional rates. This pattern suggests reduced interaction on already-correct answers rather than a uniform increase in assistance seeking. Nevertheless, the interaction rate remains higher on correct than incorrect answers, indicating that selectivity is still imperfect. Evaluation details are provided in Appendix E.2.

5 Conclusion

We presented Role-Aware Credit Assignment (RACA), which aligns training credit with specialized agent responsibilities through role-specific turn rewards, comparison groups that share the same role and reasoning state, and functionally separated advantage channels. Without a learned value function or process reward model, RACA outperforms all compared baselines on MATH Level 5 and AIME 2022–2026, improving over the strongest adaptive multi-agent method by up to 3.7 percentage points while using fewer model calls. A current limitation is that interaction rates on correct answers still exceed those on incorrect ones, leaving room for further refinement of assistance-seeking behavior.

AI USE STATEMENT

In this work, we used generative AI tools for the following tasks with required disclosure: assisting in the writing and editing of the paper text. We have not used generative AI tools for generating synthetic datasets, developing the theoretical framework or mathematical formulations, implementing parts of the training and evaluation code, formulating or proving mathematical claims, proposing or refining hypotheses, designing the research methodology or experiments, interpreting results, or translating text; these tasks were performed entirely by the authors. Qualitative and thematic data analysis is not applicable to this work.

Additionally, we used generative AI tools for the following tasks with recommended disclosure: editing the paper to improve readability, suggesting structural organization of sections.

We have reviewed all AI-assisted work. Specifically, all LLM-generated or LLM-edited text was manually reviewed, revised, and verified by the authors to ensure correctness and consistency with our intended claims. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies credit assignment for collaborative language-model reasoning on mathematical benchmarks. It does not involve human participants or the collection of personal or sensitive data, and it relies only on publicly available benchmarks and a publicly released language-model backbone. As with other language-model systems, the resulting policies may inherit biases, inaccuracies, and other limitations of the underlying model, and improved mathematical reasoning should not be interpreted as establishing reliability in safety-critical or high-stakes settings. Multi-agent reasoning and reinforcement learning also incur substantial computational costs; we therefore report inference efficiency in Sec. 4.3 and training costs in Appendix E.9. The method and resulting models should be used in accordance with applicable dataset and model licenses and responsible deployment practices.

REPRODUCIBILITY STATEMENT

We have taken several steps to make our results reproducible. The RACA method, including its role-specific rewards, comparison-group construction, and channel-routed advantage estimation, is defined in Sec. 3, with the full trajectory factorization, episode-execution rules, reward details, and optimization details given in Appendices A, B, C, and D. The backbone model, benchmarks, and shared training and evaluation protocol are described in Sec. 4.1. Data preparation and the answer extraction and matching procedure are detailed in Appendix E.1, baseline implementations in Appendix E.3, and the optimization, execution, and hardware configuration together with the complete list of reward and optimization hyperparameters in Appendices E.

REFERENCES

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate. In *International conference on learning representations*, volume 2024, pp. 9079–9093, 2024.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *Advances in Neural Information Processing Systems*, 38:46375–46408, 2026a.

Lang Feng, Longtao Zheng, Shuo He, Fuxiang Zhang, and Bo An. Dr. mas: Stable reinforcement learning for multi-agent llm systems. *arXiv preprint arXiv:2602.08847*, 2026b.

Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In Sheila A. McIlraith and Kilian Q. Weinberger (eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 2974–2982. AAAI Press, 2018. doi: 10.1609/AAAI.V32I1.11794. URL https://doi.org/10.1609/aaai.v32i1.11794.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. Vineppo: Refining credit assignment in rl training of llms. *arXiv preprint arXiv:2410.01679*, 2024.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pp. 54523–54549, 2025.

Yanming Li, Xuelin Zhang, WenJie Lu, Ziye Tang, Maodong Wu, Haotian Luo, Tongtong Wu, Zijie Peng, Hongze Mi, Yibo Feng, et al. Who deserves the reward? sharp: Shapley credit-based optimization for multi-agent system. *arXiv preprint arXiv:2602.08335*, 2026.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 17889–17904, 2024.

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.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip HS Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. Malt: Improving reasoning with multi-agent llm training. *arXiv preprint arXiv:2412.01928*, 2024.

Chanwoo Park, Seungju Han, Xingzhi Guo, Asuman E Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. Maporl: Multi-agent post-co-training for collaborative large language models with reinforcement learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 30215–30248, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Ziyu Wan, Yunxiang Li, Xiaoyu Wen, Yan Song, Hanjing Wang, Linyi Yang, Mark Schmidt, Jun Wang, Weinan Zhang, Shuyue Hu, et al. Rema: Learning to meta-think for llms with multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 38:126621–126667, 2026.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

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.

Yujie Zhao, Lanxiang Hu, Yang Wang, Minmin Hou, Hao Zhang, Ke Ding, and Jishen Zhao. Stronger-mas: Multi-agent reinforcement learning for collaborative llms. In *International Conference on Learning Representations*, volume 2026, pp. 150619–150651, 2026.

A TRAJECTORY FACTORIZATION

Let $\mathcal{C}(\tau)$ denote all controller turns in trajectory τ, and let $\mathcal{A}(\tau) \subseteq \mathcal{C}(\tau)$ denote those whose executed decision is `continue`. For each $t \in \mathcal{A}(\tau)$, let y_t be the primary proposer response, H_t the number of budgeted interaction hops, and $z_{t,h}$ the response produced by role $k_{t,h}$ at hop h. Auxiliary verifier turns outside the hop budget form $\mathcal{V}_t^{\text{aux}}(\tau)$, with responses $\widetilde{z}_{t,v}$. The trajectory factorizes as

$$\pi(\tau \mid x) = \prod_{t \in \mathcal{C}(\tau)} \pi_{\text{ctl}}(d_t \mid x, B_t) \prod_{t \in \mathcal{A}(\tau)} \left[\pi_{\text{pro}}(y_t \mid x, B_t) \prod_{h=1}^{H_t} \pi_{k_{t,h}}(z_{t,h} \mid x, B_{t,h}) \prod_{v \in \mathcal{V}_t^{\text{aux}}(\tau)} \pi_{\text{ver}}(\widetilde{z}_{t,v} \mid x, \widetilde{B}_{t,v}) \right], \tag{9}$$

where $d_t \in \{\texttt{continue}, \texttt{stop}\}$ is the controller's proposed decision, and B_t, $B_{t,h}$, and $\widetilde{B}_{t,v}$ are the blackboard states preceding the controller, budgeted response, and auxiliary verifier turns, respectively. Membership in $\mathcal{A}(\tau)$ follows the executed decision after the stop gate in Eq. (13).

The budgeted response product includes critic-triggered proposer corrections whenever the remaining hop budget permits. Auxiliary verifier turns do not contribute to H_t, but their sampled responses remain part of the policy update. Their scheduling is deterministic or independent of trainable policy parameters and therefore introduces no additional policy factor. An accepted stop contributes a controller factor but no subsequent worker factors.

B EPISODE EXECUTION DETAILS

The agents exchange information through a structured blackboard,

$$B_t = \big(\mathcal{Y}_t, \mathcal{F}_t, \mathcal{S}_t, \mathcal{I}_t\big), \tag{10}$$

where $\mathcal{Y}_t$ stores parsed proposer reasoning–answer pairs, $\mathcal{F}_t$ stores critic error descriptions, $\mathcal{S}_t$ stores verifier answer–confidence pairs, and $\mathcal{I}_t$ records interaction events. The records are append-only, so earlier contributions remain available to subsequent agents. To bound context length, each agent receives a compact natural-language rendering of this shared state rather than the complete transcript.

The context label ξ_t is recomputed at the start of every active round, before the controller acts, from the current blackboard B_t. Writing $\text{seq}(\mathcal{C})$ for the value of a single monotonically increasing insertion counter, shared across all components, at the most recent append to component $\mathcal{C}$ (with $\text{seq}(\varnothing) = -\infty$), it is set to `explore` when no solution has been written ($\mathcal{Y}_t = \varnothing$), to `refine` when the latest write is a criticism rather than a solution ($\mathcal{Y}_t \neq \varnothing$ and $\text{seq}(\mathcal{F}_t) > \text{seq}(\mathcal{Y}_t)$), and to `verify` otherwise:

$$\xi_t = \begin{cases} \texttt{explore}, & \mathcal{Y}_t = \varnothing, \\ \texttt{refine}, & \mathcal{Y}_t \neq \varnothing \text{ and } \text{seq}(\mathcal{F}_t) > \text{seq}(\mathcal{Y}_t), \\ \texttt{verify}, & \text{otherwise.} \end{cases} \tag{11}$$

These labels track the phase of collaborative reasoning—producing an initial solution, revising after criticism, and confirming a solution—and the derivation is a constant-time comparison of append positions requiring no parsing or learned classifier beyond the role-output parsing that already populates the blackboard.

When the episode terminates or exhausts its round budget, the candidate answers accumulated in $\mathcal{Y}$ determine the final answer through a uniform majority vote, with ties resolved by insertion order:

$$a = \text{mode}(\mathcal{Y}), \qquad c = \text{match}(a, a^\star). \tag{12}$$

The controller samples a proposed decision $d_t \in \{\texttt{continue}, \texttt{stop}\}$ from $\pi_{\text{ctl}}(\cdot \mid x, B_t)$. The stop gate converts this proposal into the executed decision

$$\widetilde{d}_t = \begin{cases} \texttt{stop}, & d_t = \texttt{stop} \wedge \mathcal{S}_t \neq \varnothing, \\ \texttt{continue}, & \text{otherwise.} \end{cases} \tag{13}$$

When termination is deferred, the round continues. If no verifier score is available after the budgeted interaction chain, an auxiliary verifier turn is executed before the next controller turn. An accepted stop ends the episode before another primary turn.

When the round continues, the proposer generates a reasoning trace, a candidate answer, and an interaction decision in one response:

$$(y_t, a_t, u_t, \kappa_t) \sim \pi_{\text{pro}}(\cdot \mid x, B_t), \tag{14}$$

where $u_t \in \{\texttt{none}, \texttt{request}\}$ and $\kappa_t \in \{\text{cri}, \text{ver}\}$ identifies the target when $u_t = \texttt{request}$. The reasoning trace and answer precede the interaction block in a fixed autoregressive order, so assistance-seeking is conditioned on the solution prefix already generated. The parsed reasoning–answer pair is appended to $\mathcal{Y}_t$, and any endogenous request is recorded in $\mathcal{I}_t$.

A request activates its target role using the role-specific response format. Responses update the corresponding blackboard components and may request further interaction. Each budgeted transition in this chain is a hop, and the resulting role call is a response turn. For an interaction chain, let $z_{t,h}$ be the response produced by role $k_{t,h}$ at hop h, and let $B_{t,h}$ be the blackboard state preceding that response. The update is

$$B_{t,h+1} = B_{t,h} \cup \{z_{t,h}\}, \qquad h = 1, \ldots, H_t, \quad H_t \le H, \tag{15}$$

where the union denotes appending the corresponding structured records. A critic adds an error description to $\mathcal{F}_t$ when an error is detected, a verifier adds an answer–confidence pair to $\mathcal{S}_t$, and a proposer adds a revised reasoning–answer pair to $\mathcal{Y}_t$. A detected critic error automatically schedules a proposer correction without a learned routing action. The correction executes as a response turn if hop budget remains and consumes this budget; otherwise, the criticism stays on the blackboard for a later round. Mechanically scheduled verifier-completion turns occur after the chain and are excluded from H_t.

To expose the policies to interaction outcomes when the proposer initially tends to decline assistance, training injects interactions into a subset of rounds with $u_t = \texttt{none}$. Let e_t indicate this event:

$$e_t \sim \text{Bernoulli}(\varepsilon_n), \qquad \text{if } u_t = \texttt{none}, \tag{16}$$

with

$$\varepsilon_n = \max\left(\varepsilon_{\min}, \varepsilon_0 - (\varepsilon_0 - \varepsilon_{\min})\frac{n}{n_{\max}}\right). \tag{17}$$

Here n is the optimization step, $n_{\max}$ is the maximum training budget, and ε_0 and $\varepsilon_{\min}$ are the initial injection probability and its lower bound. If a controller stop proposal has been deferred because no verifier score is available, the injected target is the verifier; otherwise, the target is sampled from the critic and verifier roles. Injected responses follow the same blackboard updates as endogenous interactions. An injected interaction is marked by $\text{forced}_t = 1$, and its responding role is trained normally. The proposer still receives solution credit, but not interaction-decision credit, because it did not choose the request. The channel mask is

$$m_t^{\text{int}} = 1 - \text{forced}_t. \tag{18}$$

The mask excludes forced turns from the interaction comparison set while retaining them in the solution channel. Merely assigning a zero interaction reward would not achieve this exclusion: the turn would still affect normalization and could receive a nonzero advantage.

During evaluation, stochastic injection is disabled, while the deterministic stop gate and verifier-completion transition remain active. This preserves the termination constraints without introducing unchosen stochastic requests.

B.1 Blackboard Rendering and Compression

The compact rendering introduced above follows fixed rules that bound prompt length:

- **Answer window.** Only the 6 most recent distinct candidate answers are shown. Each answer string is truncated to 64 characters with an ellipsis marker.
- **Flaw window.** Only the most recent critic error description is included, truncated to 600 characters; earlier descriptions are omitted.
- **Verification summary.** The answer with the highest average verification score is displayed together with its score.

- **Interaction summary.** Only the most recent interaction event (initiator, target, action type) is shown.

When one agent requests interaction, the target agent's prompt includes the requester's output, also truncated to 600 characters. These limits retain the most recent and decision-relevant information while preventing context-length overflow.

C Additional Reward Details

C.1 Proposer Correction Reward

A proposer correction triggered by the critic receives credit only for its revised answer. For response $z_{t,h}^{\mathrm{pro}}$ at hop h with parsed answer $a_{t,h}^{\mathrm{corr}}$, we use

$$r_{t,h}^{\mathrm{corr}} = \mathrm{match}\big(a_{t,h}^{\mathrm{corr}}, a^{\star}\big). \tag{19}$$

The correction may request further interaction within the remaining hop budget, but this continuation is distinct from the primary decision u_t and receives no separate interaction reward.

C.2 Critic Outcome Window

For a critic turn that correctly flags an incorrect answer, the short-horizon repair outcome is

$$q_{t,h} = \begin{cases} \widetilde{p}_t, & \text{if a proposer correction follows in round } t, \\ p_{t+1}, & \text{if the hop budget prevents a within-round correction} \\ & \text{and a later active round is executed.} \end{cases} \tag{20}$$

The first branch uses the correctness of the candidate remaining at the end of the round. The second applies when the error is detected at the final available hop and uses the next active round's primary answer. If neither a within-round correction nor a later active round is available, $q_{t,h}$ is unavailable and the critic receives b_{terminal}.

This window associates criticism with a nearby observable correction outcome rather than broadcasting final episode correctness to every critic turn. It does not isolate the causal effect of the criticism, since no otherwise identical trajectory without that criticism is observed.

C.3 Controller Reward Attachment

The controller reward is attached to the accepted stop turn. If the episode reaches the round limit without an accepted stop, it is attached to the final controller turn. Controller rewards are then aggregated at the episode level, and the resulting advantage is assigned to every sampled controller proposal because each proposal affects the subsequent continuation and verification path. Proposer, critic, and verifier rewards remain associated with their individual turns.

D Additional Optimization Details

D.1 Comparison-Group Statistics

Within each worker comparison group, turns sharing the same trajectory, round, role, response status, and reward value contribute one representative to the group statistics. The resulting advantage is assigned to every turn represented by that entry. Deduplication therefore changes the statistical weight of repeated outcomes, not their eligibility for the policy update.

A channel is retained only when its group contains at least two representatives and their reward standard deviation exceeds δ. These representatives may come from different trajectories or different rounds of the same trajectory. An omitted channel contributes neither a surrogate term nor an additional turn to the objective's normalization count. If another channel of the same turn remains valid, that turn is still included once in M.

D.2 Token-Level Probability Ratio

For token l of turn j in trajectory i, the importance ratio is

$$\rho_{j,l}^{i} = \frac{\pi_{k(j)}\big(y_{j,l}^{i} \mid s_{j}^{i}, y_{j,<l}^{i}\big)}{\pi_{k(j)}^{\mathrm{old}}\big(y_{j,l}^{i} \mid s_{j}^{i}, y_{j,<l}^{i}\big)}. \tag{21}$$

Here $\pi_{k(j)}^{\mathrm{old}}$ is the role policy used to sample the rollout, and both policies are evaluated on the same prompt and generated prefix.

D.3 Role-Specific Reference Policies and KL Estimation

For each role $k \in \{\mathrm{ctl}, \mathrm{pro}, \mathrm{cri}, \mathrm{ver}\}$, the trainable adapter is initialized from its corresponding role-specific SFT checkpoint. We maintain a separate frozen copy of that checkpoint as the reference adapter throughout reinforcement learning:

$$\Delta\theta_{k}^{(0)} = \Delta\theta_{k}^{\mathrm{SFT}}, \qquad \Delta\theta_{k}^{\mathrm{ref}} = \mathrm{copy}\big(\Delta\theta_{k}^{\mathrm{SFT}}\big).$$

The reference policy for role k is

$$\pi_{k}^{\mathrm{ref}} = \pi_{\theta_0 + \Delta\theta_{k}^{\mathrm{ref}}}, \tag{22}$$

where θ_0 denotes the shared frozen backbone parameters. Each reference adapter is held fixed at its role-specific SFT weights, while the corresponding trainable adapter is updated during reinforcement learning.

For token l of turn j in trajectory i, the nonnegative per-token KL estimator is

$$\widehat{D}_{j,l}^{i} = \exp\big(\Delta_{j,l}^{i}\big) - \Delta_{j,l}^{i} - 1, \qquad \Delta_{j,l}^{i} = \log \pi_{k(j)}^{\mathrm{ref}}\big(y_{j,l}^{i} \mid \cdot\big) - \log \pi_{k(j)}\big(y_{j,l}^{i} \mid \cdot\big). \tag{23}$$

Both policies are evaluated on the same prompt and generated prefix. This penalty constrains each role policy relative to its own SFT initialization rather than the unadapted backbone or another role's checkpoint. During optimization, turn j activates and updates only $\Delta\theta_{k(j)}$; the shared backbone and all reference adapters remain frozen.

E Additional Experimental Details

E.1 Data Preparation and Evaluation

After excluding supervised fine-tuning examples and overlaps with the evaluation sets, 5,265 problems from Levels 3–5 of the MATH training split remain and are divided into disjoint training and validation partitions. The MATH test set is a fixed set of 1,000 Level-5 problems, targeting its most challenging level. The AIME benchmark contains all problems from AIME I and AIME II between 2022 and 2026, with 30 problems per year. No AIME problem appears in the supervised fine-tuning or reinforcement-learning data. Neither test set is inspected during training or used for early stopping, checkpoint selection, or hyperparameter tuning. The selected checkpoint is evaluated once on each test set using the same answer extraction and matching procedure across methods. The reinforcement-learning training set is derived from the MATH training split by removing (i) problems whose reference answer is empty or contains unbalanced braces, which lack a verifiable ground truth; (ii) problems appearing in the supervised fine-tuning data, to prevent reward-signal contamination from memorized solutions; and (iii) problems appearing in the evaluation sets. The validation partition is a fixed subset of 300 problems, held out from the training set and disjoint from the 1,000-problem MATH test set. It is used only for convergence monitoring and checkpoint selection and is evaluated every 10 optimization steps; no validation problem is used for gradient updates.

E.2 Learning Dynamics Evaluation

Figure 3 uses a fixed MATH validation set, with accuracy and interaction statistics at each checkpoint computed from the same evaluation run. RACA's interaction curves cover 90 training steps. Interaction rates are measured at the reasoning-round level and count proposer-initiated requests, excluding mechanically scheduled calls. Correctness is determined from the primary answer before interaction. We report the overall interaction rate together with $P(\mathrm{Int} \mid \mathrm{Wrong})$ and $P(\mathrm{Int} \mid \mathrm{Correct})$.

E.3 Baseline Implementation Details

SFT-CoT uses greedy chain-of-thought decoding from the SFT checkpoint. Self-Consistency samples eight independent trajectories from the same checkpoint, while Single-Agent GRPO + Self-Consistency applies this procedure to the trained policy, providing a call-matched single-agent baseline. Self-Refine generates an initial solution and self-feedback before performing one revision. The Fixed Four-Role Pipeline follows a predetermined sequence of controller, proposer, critic, and verifier calls rather than learned routing.

Single-Agent GRPO optimizes a proposer-only policy using the final-answer reward. We evaluate both SFT and outcome-GRPO-trained variants of the Fixed Four-Role Pipeline. Within the adaptive blackboard framework, Multi-Agent Outcome GRPO broadcasts the standardized terminal outcome to every generated turn, whereas Role-Wise Reward GRPO assigns local rewards according to each role's immediate responsibility.

Unless otherwise specified, trainable RL methods share the training and evaluation settings, differing only in reasoning workflow or credit assignment.

E.4 Role-Specific Adapters

The roles share a frozen language-model backbone π_{θ_0}, with each role implemented by a separate trainable low-rank adapter $\Delta\theta_k$:

$$\pi_k = \pi_{\theta_0+\Delta\theta_k}, \qquad k \in \{\mathrm{ctl}, \mathrm{pro}, \mathrm{cri}, \mathrm{ver}\}. \tag{24}$$

Only the active role's adapter is used for each call, allowing the agents to share pretrained representations while maintaining disjoint trainable parameters. All role adapters use rank 16. The single-agent baseline activates and optimizes only the proposer adapter.

E.5 Supervised Fine-Tuning

Before reinforcement learning, we separately fine-tune four role-specific LoRA adapters for the controller, proposer, critic, and verifier over a shared frozen backbone. Each adapter is trained only on demonstration turns corresponding to its role.

The SFT dataset consists of expert episodes, each containing multiple role turns with structured system, user, and assistant fields. After removing turns whose structured output cannot be parsed (17 proposer turns missing the answer label, 12 critic turns missing the error-analysis label, and 15 verifier turns missing the score label), the dataset contains 2,870 turns across all four roles. These turns are partitioned by role to train the corresponding adapters.

For role $k \in \{\mathrm{ctl}, \mathrm{pro}, \mathrm{cri}, \mathrm{ver}\}$, we minimize the standard causal language-modeling loss over response tokens, with prompt tokens masked:

$$\mathcal{L}_{\mathrm{SFT}}^{(k)} = -\frac{1}{|\mathcal{R}_k|} \sum_{l \in \mathcal{R}_k} \log \pi_{\theta_0+\Delta\theta_k}\big(y_l \mid s_l, y_{<l}\big), \tag{25}$$

where θ_0 denotes the frozen backbone parameters, $\Delta\theta_k$ denotes the adapter parameters for role k, $\mathcal{R}_k$ contains the response-token positions in the role-k training batch, and s_l and $y_{<l}$ denote the prompt and response prefix associated with token l. Each training example updates only its corresponding role adapter; the backbone remains frozen throughout SFT.

Each role adapter is trained using AdamW with a learning rate of 2×10^{-5}, a batch size of 2, and 3 epochs. The maximum sequence length is 1,536 tokens, with gradient checkpointing enabled to fit training within a single GPU.

The resulting four role-specific SFT checkpoints initialize the corresponding adapters for reinforcement learning. Across all compared methods, each role uses the same corresponding SFT checkpoint. Single-agent methods use only the proposer checkpoint, whereas four-role methods use all four checkpoints.

E.6 Role System Prompts

Each role receives a fixed system prompt specifying its responsibilities and required output format. All prompts are in Chinese, matching the language of the demonstration data. English translations are provided below; the original Chinese prompts are included in the released code.

Controller. "You are the meta-cognitive terminator of a math reasoning team. Each round, analyze the current blackboard state and decide whether to stop. Output: `<meta-plan> decision: [continue|stop] reason: [...] </meta-plan>`. `continue`: the current answer is not yet reliable (no candidate, unresolved criticism, or low/missing verification score). `stop`: verification scores support the current answer with sufficient confidence. Do not stop when no verification score is available."

Proposer. "You are a math problem-solving expert (Proposer). (1) Generate or improve a reasoning chain and candidate answer; (2) decide whether to request assistance (assistance has communication cost; request only when genuinely needed). Output the reasoning and final answer first, then an interaction block: `<interaction> action: [none|request] target: [critic|verifier|none] reason: [...] </interaction>`. The assistance decision should be based on your confidence in the answer you have already written."

Critic. "You are a math reasoning reviewer (Critic). (1) Identify logical or computational errors; (2) decide whether to interact. Output an error analysis first (write 'No errors' if none are found), then an interaction block. When errors are found, use `request + proposer` to request a corrected solution."

Verifier. "You are an answer verification expert (Verifier). (1) Independently verify the answer and assign a confidence score in $[0, 1]$; (2) decide whether to interact. Output the score and a brief explanation first, then an interaction block. When the score is low, request improvement from the proposer or review from the critic."

E.7 Optimization and Execution Settings

Each RL update uses 32 questions with 16 sampled trajectories per question. Each response is limited to 1,280 tokens. Adaptive multi-agent methods use at most four active reasoning rounds, with up to three interaction hops per round.

The learning rate is 1×10^{-5} and the clipping coefficient is 0.2. Validation is performed before reinforcement learning and every 10 optimization steps thereafter. Only the MATH validation partition is used for convergence monitoring and checkpoint selection.

Training trajectories are sampled at temperature 1.0. At test time, decoding is greedy except for the two Self-Consistency variants, which sample eight independent trajectories at temperature 0.7 and use majority voting to select the final answer.

All training runs use a single node with eight NVIDIA A100 80 GB GPUs.

E.8 Reward and Optimization Hyperparameters

Table 5 lists the reward and optimization hyperparameters used in all experiments. All values are shared across the RACA and ablation runs unless stated otherwise.

Table 6 examines sensitivity to the principal reward and optimization hyperparameters. Each block varies one hyperparameter (or one coupled group) while holding all others at their default values (Table 5).

E.9 Training Cost

Table 7 summarizes trainable parameter counts and reinforcement-learning costs. Trainable parameters include all optimized role adapters and exclude the frozen backbone and reference adapters. Rollout tokens count generated output tokens across all roles during training. GPU-hours include

Table 5: Reward and optimization hyperparameters.

Hyperparameter	**Value**
Interaction gain g	0.3
Interaction cost c_{int}	0.02
Over-interaction penalty o	0.05
Missed-interaction penalty m	0.1
Interaction weight λ_{int}	1.0
Critic detection reward b_{tp}	0.1
Critic repair bonus g_{cri}	0.2
False-positive penalty η_{fp}	0.2
True-negative reward b_{tn}	0.1
Terminal critic reward b_{terminal}	0.2
Controller coefficient α	0.3
Controller coefficient β	0.2
Controller coefficient γ	0.3
Variance threshold δ	1×10^{-4}
Initial injection probability ε_0	0.3
Minimum injection probability $\varepsilon_{\min}$	0.25
Maximum training steps $n_{\max}$	90
KL coefficient κ	0.04

Table 6: Sensitivity to principal reward and optimization hyperparameters on MATH Level 5. Each block varies one hyperparameter while holding all others at their default values (Table 5). Results report mean $\pm$ standard deviation over three independent training runs. The default configuration (bold) corresponds to the main experiments.

Group	**Setting**	**MATH-L5** $\uparrow$
Interaction gain g	0.2	63.8 ± 0.7
	0.3 (default)	$\mathbf{64.5 \pm 0.6}$
	0.4	63.9 ± 0.8
Interaction cost c_{int}	0.01	64.1 ± 0.5
	0.02 (default)	$\mathbf{64.5 \pm 0.6}$
	0.04	63.6 ± 0.9
Missed-interaction m	0.05	63.7 ± 0.8
	0.1 (default)	$\mathbf{64.5 \pm 0.6}$
	0.2	64.0 ± 0.7
Controller (α, β, γ)	$(0.2,\ 0.1,\ 0.2)$	63.5 ± 0.9
	$(\mathbf{0.3},\ \mathbf{0.2},\ \mathbf{0.3})$ **(default)**	$\mathbf{64.5 \pm 0.6}$
	$(0.4,\ 0.3,\ 0.4)$	63.8 ± 0.7
KL coefficient κ	0.02	64.2 ± 0.7
	0.04 (default)	$\mathbf{64.5 \pm 0.6}$
	0.08	63.4 ± 0.8
Injection prob. ε_0	0.25	64.0 ± 0.6
	0.30 (default)	$\mathbf{64.5 \pm 0.6}$
	0.40	63.7 ± 0.9

rollout generation, policy optimization, and validation across all allocated GPUs, excluding the shared SFT initialization and final test evaluation. Training costs are averaged over three independent runs.

Table 7: Trainable parameter counts and reinforcement-learning costs. Token counts and GPU-hours are cumulative per run and averaged across three independent training runs.

Method	**Trainable Params (M)**	**Rollout Tokens (M)**	**GPU-hours**
Single-Agent GRPO	7.67	17.20	37.10
Fixed Four-Role + GRPO	30.67	43.00	175.41
Multi-Agent Outcome GRPO	30.67	60.96	251.39
Role-Wise Reward GRPO	30.67	72.39	318.60
AT-GRPO	30.67	83.67	349.52
GiGPO	30.67	63.68	265.45
RACA (Ours)	30.67	62.71	294.64

F Additional Baseline Analysis

Table 1 also shows the contributions of single-agent training and test-time scaling. Single-Agent GRPO exceeds SFT-CoT by 8.7 and 2.6 percentage points on MATH Level 5 and AIME, respectively. Applying $N = 8$ Self-Consistency further raises accuracy to 62.4% and 14.4%, providing a stronger baseline than scaling the SFT checkpoint alone. In contrast, the SFT-only Fixed Four-Role Pipeline improves over SFT-CoT by only 2.4 percentage points on MATH and yields no improvement on AIME, showing that a fixed composition of specialized roles does not by itself guarantee substantial gains.