LEARNING ACROSS ROLES: ROLE-BALANCED SHARED-POLICY OPTIMIZATION FOR MULTI-AGENT CODE GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Shared-policy multi-agent learning enables collaborative code generation without maintaining a separate policy for each role. However, common actor-loss reductions implicitly assign greater weight to roles that generate longer responses or are invoked more frequently, coupling optimization to workflow exposure. We propose Role-Balanced Shared-Policy Optimization (RBPO), which jointly trains a Planner, a Coder, and a Reflector through a single role-conditioned policy in an execution-guided refinement loop. RBPO averages token losses within each response and responses within each role, then combines the resulting role objectives with explicit weights. This hierarchical objective makes each represented role's nominal weight independent of its response lengths and invocation count, without adding role-specific policy parameters or rollout calls. On LiveCodeBench (LCB-v6), RBPO with Qwen3-8B achieves 60.7% accuracy, improving over Qwen3-8B GRPO by 9.8 percentage points and the same workflow without reinforcement learning by 10.4 points. It also exceeds a strong learning-based multi-agent method using Qwen3-14B by 1.3 points under the evaluated inference configurations. Controlled ablations support the contribution of both sequence and role balancing, while role-removal analyses demonstrate the importance of reflection despite its lower invocation frequency. These results highlight explicit role weighting as an effective design choice for shared-policy multi-agent code generation.

1 INTRODUCTION

Multi-agent code generation distributes problem solving across planning, implementation, and revision. When these roles share one language-model policy, they can be trained jointly without maintaining a separate policy for each role. Shared parameters, however, do not specify how the training objective combines their experience. In an execution-guided workflow, a Coder may generate long programs at every round, while a Reflector is called only after failure. Averaging their tokens or responses uniformly therefore gives different roles different objective weights, even when those weights were never chosen explicitly.

This distinction matters because workflow exposure need not reflect functional value. In our refinement loop, planning and coding occur in every round, while reflection occurs only between unsuccessful attempts. For complete trajectories with at most three rounds, reflection therefore accounts for at most one quarter of role invocations. Averaging losses across responses passes this structural asymmetry directly into the role weights; token pooling additionally depends on response lengths. We ask whether choosing the role distribution explicitly improves learning within an otherwise fixed workflow.

Prior work establishes that collaborative policies can be learned jointly and that parameter sharing has benefits and tradeoffs. CURE co-trains code and test generation in one model (Wang et al., 2025), and Stronger-MAS supports on-policy training with shared or separate role policies (Zhao et al., 2025). Dr. MAS addresses differences in agents' reward statistics through agent-wise advantage normalization (Feng et al., 2026). Most closely, Zeng et al. (2026) analyze how workflow asymmetries can concentrate training pressure on particular roles. Motivated by these findings, we

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.

study a specific intervention: explicitly choosing the role marginal under which the shared actor loss is averaged, after advantages have been computed.

We propose *Role-Balanced Shared-Policy Optimization* (RBPO). RBPO averages token losses within each response, averages responses within each role, and mixes the resulting role objectives using prescribed weights. The resulting loss is equivalent to selecting a represented role first, then one of its responses, then one generated token. This gives each represented role a fixed nominal share of the actor objective. It does not equalize gradient norms or establish optimal credit assignment; it removes an otherwise implicit dependence of role weights on response length and invocation count.

Our evidence follows the distinction between exposure, weighting, and utility. First, the workflow analysis identifies a structural source of unequal role exposure. Second, a controlled 2×2 ablation on DeepCoder changes only the two averaging levels: full RBPO reaches 30.1%, compared with 29.4% without sequence balancing, 27.4% without role balancing, and 26.3% without either. Both components improve accuracy whether the other is present or absent. Third, inference-time role removal shows that the less frequently invoked Reflector remains important for final correctness. The complete Qwen3-8B system also reaches 60.7% on LCB-v6, 10.4 percentage points above the same workflow without reinforcement learning.

Our contributions are:

- A workflow-level analysis linking early stopping to implicit role weights, and a characterization of the distinct measures induced by token, response, and role averaging.
- A shared-policy actor objective that implements a prescribed role distribution with equal within-role response weights, independently of observed token volume and invocation frequency.
- A controlled code-generation study separating loss aggregation, functional role use, and complete-system performance, with a 3.8-point accuracy gain over token pooling in the fixed-workflow comparison.

2 Related Work

Code generation through collaboration. Frozen multi-agent systems improve inference through role design and orchestration: AgentVerse organizes expert collaboration (Chen et al., 2024); ChatDev and MetaGPT model software-development workflows (Qian et al., 2024; Hong et al., 2024); MapCoder combines retrieval, planning, coding, and debugging (Islam et al., 2024); and DyLAN and AFlow adapt teams or workflows (Liu et al., 2023; Zhang et al., 2025). CodeRL and RLEF instead train individual code policies using correctness and execution feedback (Le et al., 2022; Gehring et al., 2025). These directions motivate combining learned policies with interactive execution feedback.

Learning collaborative policies. MAPoRL and MAGRPO optimize collaboration with multi-agent reinforcement learning (Park et al., 2025; Liu et al., 2025). CURE shares parameters across coding and unit-test generation (Wang et al., 2025), while CoRe-Code trains cooperating planning and coding policies (Dou et al., 2026). MARS^2 combines multi-agent reinforcement learning with tree search (Wang et al., 2026). These methods establish shared-policy training and joint optimization. RBPO studies the empirical distribution used to reduce heterogeneous role losses into one actor objective.

Advantage estimation and role weighting. AT-GRPO groups samples by agent and turn (Zhao et al., 2025), while Dr. MAS normalizes advantages using agent-specific reward statistics (Feng et al., 2026). These mechanisms alter the learning signal inside each sample loss. RBPO changes the coefficients used to aggregate those losses. The role-capture analysis of Zeng et al. (2026) connects workflow asymmetry to training pressure; we examine a targeted intervention at the aggregation stage. Hierarchical averaging itself is a familiar operation. Our focus is its role in a policy-dependent workflow, where early stopping changes the observed role mixture, and the empirical consequence of choosing the role marginal explicitly.

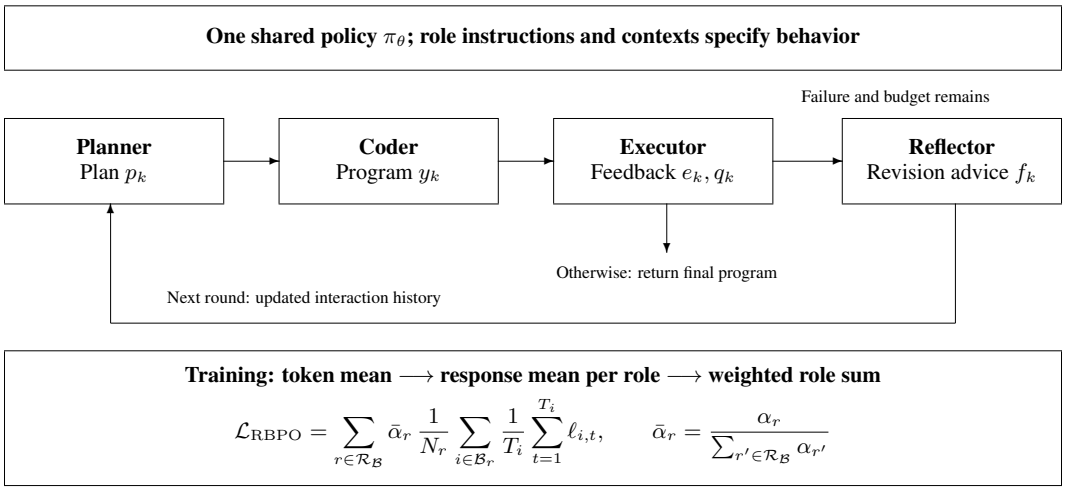


Figure 1: RBPO uses one role-conditioned policy for planning, coding, and reflection. Execution feedback determines whether another revision is attempted. During training, actor losses are averaged over tokens within responses and over responses within roles, then combined with explicit role weights. Reflection is omitted after visible-test success and after the final code-generation round.

3 METHOD

RBPO consists of a role-conditioned collaborative rollout, execution-aligned response rewards, and a hierarchical actor objective (Figure 1). The workflow is used during training and inference; role balancing changes only the training objective.

3.1 ROLE-CONDITIONED COLLABORATIVE ROLLOUTS

Given a programming problem x, a set of visible tests $\mathcal{V}$, and an execution environment $\mathcal{E}$, we consider three functional roles,

$$\mathcal{R} = \{\mathrm{Planner}, \mathrm{Coder}, \mathrm{Reflector}\}. \tag{1}$$

All roles are parameterized by the same policy π_θ. Role specialization is induced by a role instruction z_r, the role-specific context, and the required output protocol; no role-specific policy parameters are introduced.

Let h_k contain all preceding role responses and execution feedback, and let $s_k = (x, \mathcal{V}, h_k)$ denote the state at round k. Rollouts are sampled from a frozen behavior policy $\pi_{\theta_{\mathrm{old}}}$. At each round, the Planner proposes a plan, the Coder produces a candidate program conditioned on that plan, and the executor returns structured feedback together with the visible-test pass rate:

$$p_k \sim \pi_{\theta_{\mathrm{old}}}(\cdot \mid z_{\mathrm{P}}, s_k), \tag{2}$$

$$y_k \sim \pi_{\theta_{\mathrm{old}}}(\cdot \mid z_{\mathrm{C}}, s_k, p_k), \tag{3}$$

$$(e_k, q_k) = \mathcal{E}(x, y_k, \mathcal{V}), \qquad q_k \in [0, 1]. \tag{4}$$

The interaction terminates immediately if $q_k = 1$ or if $k = M - 1$, where M is the maximum number of code-generation rounds. Otherwise, the Reflector diagnoses the current attempt,

$$f_k \sim \pi_{\theta_{\mathrm{old}}}(\cdot \mid z_{\mathrm{R}}, s_k, p_k, y_k, e_k), \tag{5}$$

and the next state is formed as

$$h_{k+1} = h_k \oplus (p_k, y_k, e_k, f_k). \tag{6}$$

Thus, a complete trajectory with $K_\tau \leq M$ executed programs contains K_τ Planner responses, K_τ Coder responses, and $K_\tau - 1$ Reflector responses. In particular, no Reflector is generated after

a successful program or after the final execution budget is consumed. Complete trajectories are collected before updating the shared parameters, so every role learns on states induced by the current behaviors of its collaborators. The role signals update the same parameters, with specialization induced by their different contexts and output protocols.

How early stopping enters the objective. Consider J complete, unfiltered trajectories, with $L = \sum_{\tau=1}^{J} K_\tau$ executed programs and $\bar{K} = L/J$. They contain L Planner, L Coder, and $L - J$ Reflector responses. Under response pooling, the Reflector's nominal role mass is therefore

$$Q_{\text{resp}}(\text{R}) = \frac{L - J}{3L - J} = \frac{\bar{K} - 1}{3\bar{K} - 1} \leq \frac{M - 1}{3M - 1}. \tag{7}$$

For $M = 3$, this mass is at most $1/4$ and decreases as the mean trajectory length decreases. Thus, even equal-length responses do not give equal role weights: stopping decisions induce a role distribution inside the actor objective. This is a counting property of intact trajectories, not an empirical gradient claim. Appendix K gives the derivation and boundary cases.

3.2 Execution-Based Role-Aligned Rewards

Execution feedback provides a common, automatically verifiable measure of progress. Let $q_{-1} = 0$ when no initial program is provided, and define the improvement of the k-th program as

$$\Delta q_k = q_k - q_{k-1}. \tag{8}$$

The plan and code at round k jointly determine the program evaluated at that round, so both receive its immediate improvement:

$$r_k^{\text{P}} = r_k^{\text{C}} = \Delta q_k. \tag{9}$$

A Reflector response is generated only when a subsequent round will be attempted. Its feedback is therefore paired with the improvement of the next executed program:

$$r_k^{\text{R}} = \Delta q_{k+1}, \qquad 0 \leq k < K_\tau - 1. \tag{10}$$

The assignment aligns each response with an observed execution transition. In particular, Δq_{k+1} trains both the reflection that motivates a revision and the subsequent plan and code that realize it. This assignment is not a causal decomposition of marginal contribution; it is a tractable process reward that requires neither role-specific annotations nor counterfactual rollouts.

For roles that must follow a machine-checkable output protocol, we optionally augment the execution reward with a structural-validity term,

$$u_i = \widetilde{r}_i + \lambda_{\text{fmt}} \phi_i, \qquad \phi_i \in \{0, 1\}, \tag{11}$$

where i indexes a generated response, $\lambda_{\text{fmt}} \geq 0$, and ϕ_i indicates whether that response satisfies its role-specific format. The scalar $\widetilde{r}_i$ is the execution reward assigned above; it is distinct from the role label r_i used below. We use PPO: the assigned reward is treated as the terminal reward of response i, and the standard value baseline converts it into a stop-gradient response-level advantage $\widehat{A}_i$, which is shared by the generated tokens of that response. RBPO changes only the aggregation of the resulting PPO actor losses; the value estimator, reward normalization, probability ratio, and clipping rule are otherwise unchanged.

3.3 Role-Balanced Shared-Policy Optimization

Let an on-policy rollout batch $\mathcal{B}$ contain generated responses indexed by i. Response i is produced by role r_i, contains T_i tokens $a_{i,1:T_i}$, and has token context $c_{i,t}$. For the standard clipped PPO actor update Schulman et al. (2017), define the token-level probability ratio and loss as

$$\rho_{i,t}(\theta) = \frac{\pi_\theta(a_{i,t} \mid c_{i,t})}{\pi_{\theta_{\text{old}}}(a_{i,t} \mid c_{i,t})}, \tag{12}$$

$$\ell_{i,t}(\theta) = -\min\Big(\rho_{i,t}(\theta)\widehat{A}_i, \text{clip}(\rho_{i,t}(\theta), 1 - \epsilon, 1 + \epsilon)\widehat{A}_i\Big). \tag{13}$$

If a token-decomposable actor regularizer is used, the same aggregation can be applied to it; the invariance below concerns only terms aggregated in this way.

Table 1: The empirical measure chosen by each loss reduction. $W(i)$ sums token coefficients for response i; $Q(r)$ sums them for role r. Both middle rows are controlled ablations of RBPO.

Reduction	Role mass $Q(r)$	Response mass $W(i)$, $i \in \mathcal{B}_r$
Token pooling	S_r/S	T_i/S
Response pooling (w/o Role Bal.)	N_r/B	$1/B$
Role-wise token pooling (w/o Seq. Bal.)	$\bar{\alpha}_r$	$\bar{\alpha}_r T_i/S_r$
RBPO	$\bar{\alpha}_r$	$\bar{\alpha}_r/N_r$

RBPO first treats each role invocation as one optimization unit by averaging over the tokens in a response:

$$\mathcal{L}_i^{\text{seq}} = \frac{1}{T_i} \sum_{t=1}^{T_i} \ell_{i,t}(\theta). \tag{14}$$

For role r, let $\mathcal{B}_r = \{i \in \mathcal{B} : r_i = r\}$ and $N_r = |\mathcal{B}_r|$. The role objective is the mean over its observed invocations,

$$\mathcal{L}_r = \frac{1}{N_r} \sum_{i \in \mathcal{B}_r} \mathcal{L}_i^{\text{seq}}. \tag{15}$$

Finally, let $\alpha_r > 0$ be an explicit role weight with $\sum_{r \in \mathcal{R}} \alpha_r = 1$, and let $\mathcal{R}_{\mathcal{B}} = \{r : N_r > 0\}$ be the roles represented in the rollout batch. The RBPO actor objective is

$$\mathcal{L}_{\text{RBPO}} = \sum_{r \in \mathcal{R}_{\mathcal{B}}} \frac{\alpha_r}{\sum_{r' \in \mathcal{R}_{\mathcal{B}}} \alpha_{r'}} \mathcal{L}_r. \tag{16}$$

Only represented roles enter the objective. Write $\bar{\alpha}_r = \alpha_r / \sum_{r' \in \mathcal{R}_{\mathcal{B}}} \alpha_{r'}$; if reflection is absent, its weight is redistributed over Planner and Coder. The counts and normalization refer to the complete rollout batch and are retained during minibatch optimization (Appendix I).

Choosing the target distribution. We use uniform α_r as a symmetric default: each represented role receives the same nominal share regardless of how often the workflow calls it. This is a macro-average over roles, in contrast to a micro-average over their observed responses or tokens. It expresses a training preference, not an assumption that the roles have equal rewards, gradient norms, or optimal importance. Nonuniform preferences can be specified by changing α without changing the workflow.

Table 1 compares the induced measures. Let $S_r = \sum_{i \in \mathcal{B}_r} T_i$, $S = \sum_r S_r$, and $B = |\mathcal{B}|$. RBPO is equivalently the expectation under $Q_{\mathcal{B}}(r, i, t) = \bar{\alpha}_r/(N_r T_i)$: select a represented role using $\bar{\alpha}$, then a response uniformly within that role, then a generated token uniformly within that response.

Property 1 (Characterization of the aggregation rule). Fix the represented roles and their prescribed masses $\bar{\alpha}_r$. Among normalized linear reductions of token losses, requiring equal token coefficients within each response and equal total response coefficients within each role uniquely gives

$$w_{i,t} = \frac{\bar{\alpha}_{r_i}}{N_{r_i} T_i}, \qquad \sum_{i \in \mathcal{B}_r} \sum_{t=1}^{T_i} w_{i,t} = \bar{\alpha}_r. \tag{17}$$

To see this, equal response masses in role r must each be $\bar{\alpha}_r/N_r$; distributing that mass equally over the T_i tokens fixes $w_{i,t}$. Conversely, these weights satisfy both requirements and the prescribed role marginal. Hence the role coefficient is independent of token volume and invocation count, conditional on representation. The characterization explains the objective's design constraints; it is not an optimality result.

At fixed rollout data, the gradient is $\nabla_\theta \mathcal{L}_{\text{RBPO}} = \sum_{r \in \mathcal{R}_{\mathcal{B}}} \bar{\alpha}_r \nabla_\theta \mathcal{L}_r$. These scalar mixture weights are controlled, while the role gradients can differ in magnitude and direction. RBPO neither restores experience from absent roles nor removes failure-conditioned sampling of reflection. It changes how observed experience is weighted. The aggregation adds no policy parameters, role calls, or search branches to a fixed rollout procedure.

4 EXPERIMENTS

Our central experimental question is whether choosing the actor-loss reduction improves learning within a fixed shared-policy workflow. We address it with a four-way objective ablation, alongside system-level comparisons and inference-time role interventions. All reported endpoint results use the final training checkpoint under a common selection rule.

4.1 EXPERIMENTAL SETUP

Data and metric. Following MARS^2 (Wang et al., 2026), training uses the open-source code-generation data released with DeepCoder (Luo et al., 2025). We report main results on the evaluated LCB-v6 set (Jain et al., 2025) and ablations on the DeepCoder test set. Training and evaluation data are strictly disjoint, and the evaluation sets are kept separate. Accuracy is the percentage of problems whose final submitted program passes all evaluation tests. Each run returns one final program, even when it contains several intermediate attempts. Across evaluations, refinement and early stopping use only public examples or separately designated visible tests. Final accuracy is computed using disjoint hidden tests, whose outcomes are never fed back to the policy. The visible-test pass rate q_k is therefore a workflow signal, not final benchmark correctness. All experimental results are reported as means over two runs with different random seeds per configuration.

Comparisons and their scope. We rerun every method in Table 4; all entries are results of our own experiments. Single-agent controls comprise direct Qwen3-8B/14B generation (Yang et al., 2025) and Qwen3-8B trained with PPO or GRPO (Schulman et al., 2017; Shao et al., 2024). Frozen systems include DyLAN, AgentVerse, MetaGPT, MapCoder, and AFlow. We reproduce these systems as Qwen3-8B adaptations in a common evaluation harness, aligning the task representation, visible-test interface, output protocol, sandbox, decoding configuration, context limits, correctness checker, and prompts for shared functional roles. Their orchestration and routing differ, so one run does not imply an equal token budget. These are comparisons of the adapted implementations, not estimates of their original published configurations. Ours w/o RL uses the same workflow as RBPO with the untrained backbone.

MARS^2 is a trainable multi-agent reference. Our Qwen3-8B/14B runs use Pass@1(MCTS): one final program selected from a 16-node search (Wang et al., 2026). RBPO uses at most three code-generation rounds. These system-level comparisons involve different inference protocols; the factorized ablations isolate the actor aggregation within the RBPO workflow.

Training configuration. All three RBPO roles share Qwen3-8B with explicit thinking enabled. Training uses 10 epochs and 8 GPUs, generation and PPO minibatch sizes of 64, a per-GPU microbatch size of 1, and one PPO optimization epoch per rollout batch. The actor learning rate is 10^{-6}, clipping parameter 0.2, and weight decay 0.01. Rollouts use temperature 1.0, top-p 1.0, and top-$k = -1$. Prompt, response, and total context limits are 8,192, 2,048, and 10,240 tokens, respectively. We use at most three code-generation rounds, early stopping on visible-test success, and equal weights over represented roles.

4.2 DOES THE AGGREGATION RULE MATTER?

We keep the shared policy, workflow, reward, advantage estimator, PPO settings, and training budget fixed, and compare all four reductions in Table 1. Removing sequence balancing pools tokens within each role, preserving role mass but weighting that role's responses by length. Removing role balancing averages response losses globally, preserving equal response mass but weighting roles by invocation frequency. Removing both levels pools all generated-token losses across the shared workflow. Appendix B gives the full objectives.

Table 2 gives a complete 2×2 comparison of the averaging levels. Full RBPO improves on token pooling by 3.8 percentage points. The four entries also allow each component to be assessed under both settings of the other: **Role balancing** improves accuracy by 3.1 points without sequence balancing ($29.4 - 26.3$) and by 2.7 points with it ($30.1 - 27.4$). **Sequence balancing** improves accuracy by 1.1 points without role balancing ($27.4 - 26.3$) and by 0.7 points with it ($30.1 - 29.4$).

Table 2: Controlled actor-objective ablation on the DeepCoder test set with Qwen3-8B. Each entry is the mean accuracy over two training runs with different random seeds. All variants share the same Planner–Coder–Reflector workflow, reward, and training budget. Seq. and Role indicate whether the corresponding balancing level is enabled. Decrease is relative to full RBPO, in percentage points.

Actor objective	Seq.	Role	Accuracy (%)	Decrease (pp)
RBPO	✓	✓	**30.1**	—
RBPO w/o Seq. Bal.	×	✓	29.4	0.7
RBPO w/o Role Bal.	✓	×	27.4	2.7
RBPO w/o Both (token pooling)	×	×	26.3	3.8

Table 3: Inference-only role removal on DeepCoder, averaged over two runs using each run's final RBPO checkpoint. First attempt and Final are accuracies (%). Net gain is Final minus First attempt, in percentage points; it measures the net accuracy change after the first attempt. Rounds is the mean number of code-generation rounds.

Condition	First attempt	Final	Net gain	Rounds
Full workflow	21.98	30.10	8.12	2.51
No Planner	19.80	26.78	6.98	2.55
No Reflector	21.98	24.45	2.47	2.55

Thus, neither component's observed benefit depends on the other being enabled, and role balancing has the larger effect under both comparisons. These are contrasts of the reported means, not additional experimental runs or significance tests. Appendix L gives the factorial decomposition. The intervention changes only loss aggregation, making this the direct evidence for RBPO's optimization choice.

The aggregate invocation audit provides context for the three variants with available role-count statistics (Appendix C). Planner and Coder each account for approximately 39% of collected responses, and Reflector for approximately 22%. Response pooling inherits this imbalance. RBPO instead targets one third per role whenever all three are present in a batch. The audit illustrates why equal response weighting and equal role weighting lead to different objectives in this workflow. The objective masses describe loss coefficients, while the accuracy ablation measures their empirical consequence.

4.3 Do the Roles Retain Distinct Functions?

Within each run, we remove the Planner or Reflector at inference from the same final checkpoint, keeping the maximum code-generation budget at three rounds on DeepCoder (Table 3). Removing the Planner reduces first-attempt accuracy by 2.18 points and final accuracy by 3.32 points. Removing the Reflector leaves the first attempt unchanged but lowers final accuracy by 5.65 points. Equivalently, the accuracy gained after the first attempt falls from 8.12 to 2.47 points. These interventions show complementary functions: planning improves the initial solution, while reflection helps convert execution feedback into successful revisions. The less frequently invoked Reflector has the larger removal effect, motivating a distinction between invocation frequency and functional value.

Reward and interaction budget. With RBPO aggregation fixed, incremental execution rewards outperform absolute pass-rate and final-binary rewards in the reported curves (Appendix E). Of maximum round budgets $M \in \{2, 3, 4\}$, three gives the best reported final performance (Appendix F). These comparisons support incremental feedback and a three-round interaction budget for the evaluated configuration.

4.4 Complete-System Performance on LCB-v6

RBPO reaches 60.7% overall accuracy (Table 4), exceeding direct Qwen3-8B generation by 15.0 percentage points and single-agent GRPO by 9.8 points. It exceeds the Qwen3-8B and Qwen3-14B

Table 4: LCB-v6 accuracy (%), averaged over two runs with different random seeds. All methods are rerun by us; trained methods use their final checkpoints. Frozen-MAS entries are adapted Qwen3-8B implementations. MARS^2 uses 16-node search and RBPO uses at most three code-generation rounds, so inference budgets differ. "–" denotes an unreported result.

Category	Method	RL	Easy	Medium	Hard	Overall
Single agent	Qwen3-8B Direct	×	95.4	48.1	17.5	45.7
	Qwen3-14B Direct	×	–	–	–	56.0
	Qwen3-8B PPO	✓	100.0	50.0	21.3	49.1
	Qwen3-8B GRPO	✓	97.7	57.7	21.3	50.9
Frozen MAS	DyLAN	×	100.0	73.1	18.8	54.9
	AgentVerse	×	97.7	55.8	21.3	50.3
	MetaGPT	×	95.4	55.8	23.8	50.9
	MapCoder	×	83.7	26.9	1.3	29.1
	AFlow	×	95.4	59.6	20.0	50.3
Trainable MAS	MARS^2 (Qwen3-8B)	✓	97.7	61.5	26.3	54.3
	MARS^2 (Qwen3-14B)	✓	–	–	–	59.4
Shared workflow	Ours w/o RL	×	97.7	55.8	21.3	50.3
	RBPO	✓	100.0	69.2	33.8	**60.7**

MARS^2 results by 6.4 and 1.3 points, respectively, under their different inference protocols. These comparisons establish the performance of the complete system, but do not isolate role balancing from workflow, reward, or inference-budget effects.

Learning beyond orchestration. The same workflow without reinforcement learning achieves 50.3%, giving a 10.4-point improvement with RBPO training. The reported Easy, Medium, and Hard gains are 2.3, 13.4, and 12.5 points. DyLAN is stronger on Medium problems (73.1% versus 69.2%), whereas RBPO is stronger on Hard problems (33.8% versus 18.8%). The gains are therefore not uniform across difficulty levels. Figure S1 shows how accuracy evolves during training.

4.5 Parameter Sharing and Compute

RBPO deploys one 8B policy for all three roles, avoiding separate role-specific policy weights. Each planning, coding, and reflection call still incurs computation, so parameter sharing and compute savings are distinct properties. Appendix J reports approximate costs for the 8B configurations, including MARS^2, with their accounting scope. The controlled aggregation comparison requires no extra role calls or search budget by construction; its benefit is obtained by changing the training weights.

5 Discussion and Limitations

The results support a specific design principle: choose the role distribution of a shared actor objective explicitly when workflow topology creates unequal exposure. Equation 7 shows how early stopping induces that inequality; the factorial comparison shows positive effects of both averaging levels; and role removal shows why low invocation frequency need not imply low functional value. These are complementary analyses of exposure, optimization, and role use, rather than three estimates of the same effect.

The findings cover Qwen3-8B and a fixed code-generation workflow; the four-way objective ablation is evaluated on DeepCoder. Additional model families and workflows are needed to establish broader applicability. Results are means of two runs, so small differences should not be interpreted as statistically established rankings. The coefficient invariance in Property 1 does not imply equal gradient influence or optimal role weights. Role removal measures functional dependence, rather than parameter-level knowledge transfer, and system-level comparisons use different inference budgets.

Adaptive weights, matched separate-policy controls, and complete end-to-end cost measurements remain useful directions.

6 CONCLUSION

RBPO controls how heterogeneous role experience enters a shared policy update. Averaging within responses and roles removes deterministic dependence of nominal role weights on token volume and invocation count, conditional on batch representation. In a shared Planner–Coder–Reflector workflow, the reported accuracy comparisons and factorized ablations support this design choice. The broader implication is that parameter sharing and loss aggregation should be specified separately when training collaborative language-model systems.

AI USE STATEMENT

Generative AI tools assisted Chinese-to-English translation, manuscript restructuring and language editing, literature checking, methodological critique, interpretation of existing results, and clarification of mathematical notation and claims. They also assisted migration to the ICLR 2027 LaTeX template, compilation, and typesetting checks. No new experimental measurements were generated in this manuscript-editing process. The authors are responsible for verifying AI-assisted text, derivations, citations, and the consistency of all claims with the underlying experimental records, and for the final submitted content.

REPRODUCIBILITY STATEMENT

Section 3 specifies the rollout, reward alignment, and actor objective. Section 4 reports the available model and training configuration. The appendix gives factorized objectives, exposure diagnostics, additional ablations, and an implementation reference for batch weighting. All methods in the main comparison are rerun by us. Reported endpoint results use the final training checkpoint and are averaged over two runs with different random seeds. Training and evaluation data are strictly separated, as are visible feedback tests and hidden scoring tests.

REFERENCES

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/578e65cdee35d00c708d4c64bce32971-Abstract-Conference.html.

Zhihao Dou, Qinjian Zhao, Zhongwei Wan, Xiaoyu Xia, and Sumon Biswas. CoRe-Code: Collaborative reinforcement learning for code generation, 2026. URL https://arxiv.org/abs/2605.24812.

Lang Feng, Longtao Zheng, Shuo He, Fuxiang Zhang, and Bo An. Dr. MAS: Stable reinforcement learning for multi-agent LLM systems, 2026. URL https://arxiv.org/abs/2602.08847.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. RLEF: Grounding code LLMs in execution feedback with reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 19034–19055, 2025. URL https://proceedings.mlr.press/v267/gehring25a.html.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 30016–30030, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/c1e2faff6f588870935f114ebe04a3e5-Abstract-Conference.html.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/6507b115562bb0a305f1958ccc87355a-Abstract-Conference.html.

Md. Ashraful Islam, Mohammed Eunus Ali, and Md Rizwan Parvez. MapCoder: Multi-agent code generation for competitive problem solving. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4912–4944, 2024. doi: 10.18653/v1/2024.acl-long.269. URL https://aclanthology.org/2024.acl-long.269/.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 21314–21328, 2022. doi: 10.52202/068431-1549.

Shuo Liu, Tianle Chen, Zeyu Liang, Xueguang Lyu, and Christopher Amato. LLM collaboration with multi-agent reinforcement learning, 2025. URL https://arxiv.org/abs/2508.04652.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration, 2023. URL https://arxiv.org/abs/2310.02170.

Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. DeepCoder: A fully open-source 14b coder at o3-mini level. Notion Blog, 2025. URL https://pretty-radio-b75.notion.site/DeepCoder-A-Fully-Open-Source-14B-Coder-at-O3-mini-Level-1cf81902c14680b3bee5eb349a512a51.

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. doi: 10.18653/v1/2025.acl-long.1459. URL https://aclanthology.org/2025.acl-long.1459/.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, 2024. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Shijie Wang, Pengfei Li, Yikun Fu, Kaifeng Liu, Fangyuan Li, Yang Liu, Xiaowei Sun, Zonglin Li, Siyao Zhao, Jian Zhao, Kai Tian, Dong Li, Junqi Gao, Yutong Zhang, Yiqun Chen, Yuqiang Li, Zoe Li, Weinan Zhang, Peng Ye, Shuyue Hu, Lei Bai, Bowen Zhou, Kaiyan Zhang, and Biqing Qi. MARTI-MARS^2: Scaling multi-agent self-search via reinforcement learning for code generation, 2026. URL https://arxiv.org/abs/2602.07848.

Yinjie Wang, Ling Yang, Ye Tian, Ke Shen, and Mengdi Wang. Co-evolving LLM coder and unit tester via reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/d38653cdaa8e992549e1e9e1621610d7-Abstract-Conference.html.

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, 2025. URL https://arxiv.org/abs/2505.09388.

Yifan Zeng, Yiran Wu, Yaolun Zhang, Wentian Zhao, Kun Wan, Qingyun Wu, and Huazheng Wang. When does multi-agent RL improve LLM workflows? workflow, scale, and policy-sharing trade-offs, 2026. URL https://arxiv.org/abs/2605.24202.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5492ecbce4439401798dcd2c90be94cd-Abstract-Conference.html.

Yujie Zhao, Lanxiang Hu, Yang Wang, Minmin Hou, Hao Zhang, Ke Ding, and Jishen Zhao. Stronger-MAS: Multi-agent reinforcement learning for collaborative LLMs, 2025. URL https://arxiv.org/abs/2510.11062.

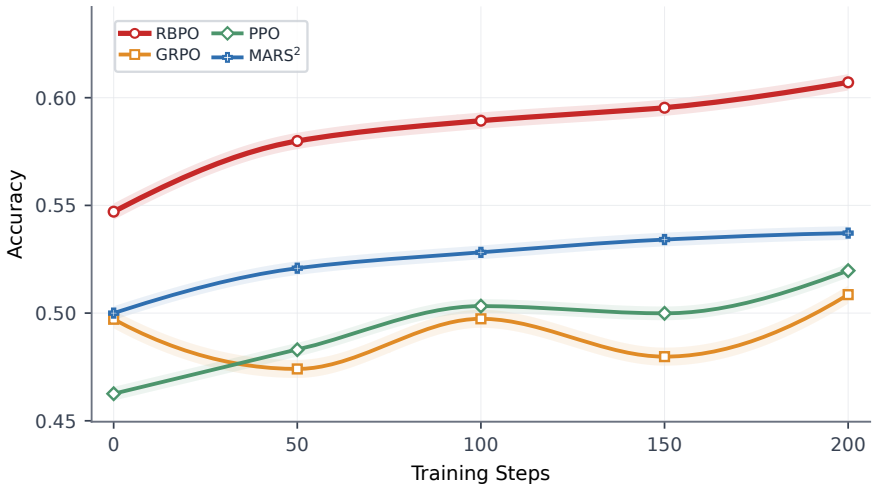


Figure S1: Mean LCB-v6 accuracy over two random seeds during training. The methods use their respective workflows. Final-checkpoint accuracy comparisons are reported in Table 4; Table 2 isolates the aggregation rule within the shared workflow.

A Supplementary Experimental Setup

This supplement provides controlled experiments that isolate the optimization and workflow components of Role-Balanced Shared-Policy Optimization (RBPO). All experimental results are means over two runs with different random seeds per configuration, consistent with the main paper. The experiments use Qwen3-8B with explicit thinking enabled, consistent with the main experimental configuration, and report endpoint accuracy using the final training checkpoint on the DeepCoder test set, following the same selection rule as the main experiments. Training and evaluation data are strictly disjoint, and the DeepCoder and LCB-v6 evaluation sets are kept separate. Visible feedback tests and hidden scoring tests are also strictly separated. Within each ablation, we keep the model initialization, role instructions, training data, rollout procedure, optimizer, decoding configuration, and evaluation harness fixed, and vary only the component named in that ablation. Figure S1 separately reports the single-agent PPO and GRPO objectives, which omit RBPO's sequence-to-role aggregation hierarchy. Those curves are conventional single-agent references; the factorized experiment below complements them by holding the complete shared-policy Planner–Coder–Reflector workflow fixed and removing either or both balancing levels.

The supplement addresses five questions. First, do sequence-level and role-level balancing remove distinct sources of exposure-induced weighting, and are both needed for the complete RBPO objective? Second, do the observed invocation exposures map to the objective-implied role masses predicted by the factorized objectives? Third, do the Planner and Reflector provide distinct functional benefits at inference time? Fourth, how much does execution-aligned incremental reward contribute relative to absolute pass-rate and final-binary rewards? Fifth, how does the maximum number of collaborative rounds affect accuracy?

B Factorized Ablation of Hierarchical Balancing

We compare four actor reductions within the same shared-policy Planner–Coder–Reflector workflow: full RBPO, RBPO without sequence balancing, RBPO without role balancing, and token pooling without either level. The role-conditioned policy, rollout procedure, reward assignment, advantage estimation, PPO clipping and value learning, optimization hyperparameters, and training budget are held fixed. Each configuration is trained twice with different random seeds. Table 2 reports the mean accuracy on the DeepCoder test set. The single-agent PPO and GRPO curves in Figure S1 are separate system-level references.

Let $\mathcal{B}$ denote a rollout batch. Response i is generated by role r_i, contains T_i generated tokens, and has token-level clipped PPO actor loss $\ell_{i,t}$. Let $\mathcal{B}_r = \{i \in \mathcal{B} : r_i = r\}$, $N_r = |\mathcal{B}_r|$, and let $\mathcal{R}_\mathcal{B}$ be the set of roles represented in the batch. Following the default setting in the main paper, we use uniform role weights; after renormalization over $\mathcal{R}_\mathcal{B}$, every represented role receives weight $1/|\mathcal{R}_\mathcal{B}|$.

Complete RBPO first averages token losses within every response, then averages responses within each role, and finally combines the represented-role objectives uniformly:

$$\mathcal{L}_{\text{RBPO}} = \frac{1}{|\mathcal{R}_\mathcal{B}|} \sum_{r \in \mathcal{R}_\mathcal{B}} \frac{1}{N_r} \sum_{i \in \mathcal{B}_r} \frac{1}{T_i} \sum_{t=1}^{T_i} \ell_{i,t}, \tag{S1}$$

Thus, every observed invocation receives equal nominal mass within its role, and every represented role receives equal nominal mass in the team objective.

RBPO w/o Seq. Bal. retains uniform role-level aggregation but pools tokens within each role without first assigning equal mass to its individual responses:

$$\mathcal{L}_r^{\text{w/o Seq}} = \frac{1}{\sum_{i \in \mathcal{B}_r} T_i} \sum_{i \in \mathcal{B}_r} \sum_{t=1}^{T_i} \ell_{i,t}, \tag{S2}$$

$$\mathcal{L}_{\text{w/o Seq}} = \frac{1}{|\mathcal{R}_\mathcal{B}|} \sum_{r \in \mathcal{R}_\mathcal{B}} \mathcal{L}_r^{\text{w/o Seq}}. \tag{S3}$$

This ablation preserves the uniform represented-role marginal, but it restores dependence on response length below the role level: within a role, the nominal mass of response i is proportional to T_i.

RBPO w/o Role Bal. retains the token-to-sequence averaging step but averages all observed responses without first grouping them by role:

$$\mathcal{L}_{\text{w/o Role}} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \frac{1}{T_i} \sum_{t=1}^{T_i} \ell_{i,t}. \tag{S4}$$

It assigns equal nominal mass to every response, but it restores dependence on invocation frequency at the role level: the nominal role marginal is $N_r/|\mathcal{B}|$.

RBPO w/o Both removes both averaging levels and pools all generated tokens in the rollout batch:

$$\mathcal{L}_{\text{w/o Both}} = \frac{1}{\sum_{j \in \mathcal{B}} T_j} \sum_{i \in \mathcal{B}} \sum_{t=1}^{T_i} \ell_{i,t}. \tag{S5}$$

Its nominal role mass is $\sum_{i \in \mathcal{B}_r} T_i / \sum_{j \in \mathcal{B}} T_j$, and its response mass is $T_i / \sum_{j \in \mathcal{B}} T_j$. This control retains the same collaborative workflow while allowing both response length and invocation frequency to influence nominal role weights.

Table 2 reports mean accuracies of 30.1% for RBPO, 29.4% without sequence balancing, 27.4% without role balancing, and 26.3% without either level. The respective decreases from full RBPO are 0.7, 2.7, and 3.8 percentage points. Retaining role balancing alone outperforms retaining sequence balancing alone in this comparison, and retaining both gives the strongest result. This ordering supports the two aggregation choices under the tested configuration; it is not a statistical significance claim or a guarantee across workflows.

Sequence balancing controls response coefficients within a role, whereas role balancing controls the marginal across represented roles. The available exposure audit in Section C covers full RBPO and the two single-removal variants; no exposure statistics are reported for the token-pooled control.

C Role Exposure and Objective-Implied Mass

To complement the controlled performance intervention in Section B, we audit the invocation-frequency exposure present in the collected trajectories. For each trained variant, let $\mathcal{C}$ denote the

Table S1: Observed invocation share and objective-implied actor-loss mass for the three factorized objectives. Each block reports mean invocation shares over two runs of the corresponding trained variant. Q_{seq} is the observed invocation share. Q_{obj} is a diagnostic objective-implied role marginal: it equals Q_{seq} for RBPO w/o Role Bal., whereas RBPO and RBPO w/o Seq. Bal. target 33.33% when all three roles are represented. The signed shift $\Delta_{\mathrm{mass}} = Q_{\mathrm{obj}} - 33.33\%$ is reported in percentage points. These quantities characterize the objective-level role weights induced by each aggregation rule.

Training objective	Role	Invocation share Q_{seq} (%)	Objective mass Q_{obj} (%)	Δ_{mass} (pp)
RBPO	Planner	39.14	33.33	0.00
	Coder	39.14	33.33	0.00
	Reflector	21.74	33.33	0.00
RBPO w/o Seq. Bal.	Planner	39.18	33.33	0.00
	Coder	39.17	33.33	0.00
	Reflector	21.65	33.33	0.00
RBPO w/o Role Bal.	Planner	39.11	39.11	$+5.78$
	Coder	39.12	39.12	$+5.79$
	Reflector	21.77	21.77	-11.56

response collection from one run used for this diagnostic, let $\mathcal{C}_r = \{i \in \mathcal{C} : r_i = r\}$, and let $n_r = |\mathcal{C}_r|$. The invocation share for that run is

$$Q_{\mathrm{seq}}(r) = \frac{n_r}{|\mathcal{C}|}. \tag{S6}$$

The table reports the mean of these shares over two runs. For an individual run, this is the response-pooled marginal over its collection and provides an aggregate diagnostic of the invocation-count dependence retained by RBPO w/o Role Bal.; the training objective applies the same response pooling separately within each rollout batch. For uniform role weights, complete RBPO and RBPO w/o Seq. Bal. instead assign, within each rollout batch,

$$Q_{\mathrm{RBPO}}(r) = \frac{1}{|\mathcal{R}_{\mathcal{B}}|}, \qquad r \in \mathcal{R}_{\mathcal{B}}. \tag{S7}$$

The two role-balanced objectives differ below the role level. For response i generated by role r_i, their nominal response masses are

$$\begin{aligned} W_{\text{w/o Seq}}(i) &= \frac{1}{|\mathcal{R}_{\mathcal{B}}|} \frac{T_i}{\sum_{j \in \mathcal{B}_{r_i}} T_j}, \\ W_{\mathrm{RBPO}}(i) &= \frac{1}{|\mathcal{R}_{\mathcal{B}}| N_{r_i}}. \end{aligned} \tag{S8}$$

Thus, RBPO w/o Seq. Bal. and complete RBPO have the same represented-role marginal, but only complete RBPO assigns equal nominal mass to observed invocations within a role. Table S1 reports the observed aggregate invocation share and the corresponding diagnostic objective mass. The Q_{obj} column selects Q_{RBPO} for the two role-balanced variants and Q_{seq} for RBPO w/o Role Bal.; the final column reports its signed deviation from the uniform three-role target.

The most stable pattern is the invocation imbalance. Across all three trained variants, Q_{seq} is nearly unchanged: Planner and Coder each account for 39.11–39.18% of invocations, whereas Reflector accounts for only 21.65–21.77%. This stable pattern is consistent with the workflow structure and is not peculiar to one trained variant: Planner and Coder participate in every attempted round, while a Reflector is generated only when another revision round will be attempted. Invocation frequency is precisely the exposure variable targeted by role balancing.

RBPO w/o Role Bal. maps that observed invocation imbalance directly to its objective-implied role marginal. Its Reflector mass is 21.77% ($\Delta_{\mathrm{mass}} = -11.56$ percentage points), while Planner and Coder receive 39.11% and 39.12% ($+5.78$ and $+5.79$ points). This coefficient shift is the one predicted for the single-removal variant with the larger performance degradation in Table 2, making

the exposure audit consistent with the controlled outcome while not treating the table itself as causal evidence.

Complete RBPO and RBPO w/o Seq. Bal. both target a 33.33% role marginal when all three roles are represented, so their distinction is intentionally absent from the role-level Q_{obj} and Δ_{mass} columns. The zero shifts for RBPO w/o Seq. Bal. do not imply that its ablation has no effect; they state only that its role-level marginal remains uniform. Its difference occurs within each role: RBPO w/o Seq. Bal. assigns invocation i a coefficient proportional to T_i, whereas complete RBPO assigns equal nominal mass to observed invocations from the same role. Because only aggregate role-level statistics are retained, Table S1 cannot estimate within-role response-length dispersion, and we therefore do not report a sequence-imbalance statistic. The sequence-level distinction is specified by the objective coefficients in Equation S8, while its empirical consequence is measured by the controlled performance gap in Table 2.

The 33.33% entries are conditional per-batch targets when all three roles are represented, while the Q_{obj} column is a diagnostic rather than an estimate of the roles' long-run nominal masses across batches. Aggregate invocation share does not identify a batch-level role-absence rate, so we make no such claim. As in the main method, uniform role weighting is conditional on representation in the batch and does not force all roles to occur equally often.

Together, Table 2 and Table S1 provide complementary evidence. Section B removes one averaging level at a time and measures the resulting performance ordering; this section audits whether the observed invocation exposure maps to the role-level coefficients predicted by those objectives. Among the two single-removal variants covered by this audit, the larger degradation occurs when the stable $39/39/22$ invocation mixture is left uncorrected, while the smaller degradation retains the uniform role marginal but restores length-dependent weighting within roles. Removing both levels yields a further accuracy decrease, although no corresponding invocation audit is available. This consistency supports the objective-level interpretation of hierarchical balancing without attributing the result to added model capacity or interaction cost.

D Inference-Time Functional Role Ablation

Within each run, we conduct an inference-only ablation of the same trained RBPO checkpoint to test whether the Planner and Reflector provide distinct functional benefits. All conditions evaluate the same DeepCoder examples using Qwen3-8B with explicit thinking enabled, the same decoding configuration, and a maximum of three code-generation rounds. The full workflow uses a Planner, a Coder, and a failure-triggered Reflector. No Planner removes the Planner response from every round and lets the Coder operate on the problem and available interaction history. No Reflector removes generated reflection while retaining prior programs and verifier feedback for subsequent rounds.

Table S2: Inference-time functional role ablation, averaged over two runs. First and Final denote first-attempt and final accuracy, and both are percentages. Net gain is Final minus First in percentage points, and Rounds is the mean number of code-generation rounds.

Condition	First	Final	Net gain	Rounds
Full	21.98	30.10	8.12	2.51
No Planner	19.80	26.78	6.98	2.55
No Reflector	21.98	24.45	2.47	2.55

Table S2 reveals a temporal division of labor. Removing the Planner reduces first-attempt accuracy by 2.18 percentage points and final accuracy by 3.32 points. The Planner's benefit is therefore already visible before the first execution, consistent with an upstream contribution to problem decomposition and initial solution formation.

Removing the Reflector produces the complementary pattern. First-attempt accuracy remains exactly 21.98%, as expected because reflection is invoked only after a failed execution and cannot affect the first attempt. Nevertheless, final accuracy falls by 5.65 points, and the accuracy gained after the first attempt decreases from 8.12 to 2.47 percentage points. The Reflector therefore contributes primarily after execution feedback by helping convert failed attempts into successful revisions. The

full workflow achieves higher final accuracy with a similar, slightly smaller mean number of code-generation rounds (2.51 versus 2.55). Thus, the observed advantage does not require a larger average number of code attempts, although total generated tokens can differ across conditions.

This result also clarifies the role-exposure analysis in Table S1. The Reflector accounts for only about 22% of observed role invocations, yet its removal produces the larger final-accuracy degradation among the two role removals. Invocation frequency is therefore not a reliable proxy for functional utility, motivating the separation of workflow frequency from role weighting. The exposure collection and inference intervention are different diagnostics and do not establish a quantitative relationship between training mass and role utility. This inference-only ablation measures sensitivity to role removal under a matched code-generation-round budget. Because role removal also changes the total number of generated responses, total-compute matching and adaptive role-weight selection remain complementary evaluation directions.

E Ablation of Execution-Based Rewards

We compare the proposed execution-aligned improvement reward with two alternatives while retaining the RBPO actor aggregation. Let $q_k \in [0, 1]$ be the visible-test pass rate of the program generated at round k. The proposed reward uses the change in pass rate, $\Delta q_k = q_k - q_{k-1}$, for the Planner and Coder, and assigns Δq_{k+1} to the Reflector that precedes the corresponding revision. The absolute pass-rate baseline replaces each incremental reward with the corresponding attained pass rate, using q_k for Planner and Coder and q_{k+1} for the preceding Reflector. It retains graded intermediate feedback but does not isolate improvement over the previous attempt. The final-binary baseline uses only final trajectory correctness, $\mathbb{I}[q_{K_\tau - 1} = 1]$, and therefore removes both partial-progress information and intermediate execution supervision.

As shown in Figure S2, the proposed reward achieves the strongest final performance, followed by the absolute pass-rate reward and the final-binary reward. The comparison separates reward design from actor-loss balancing because all three configurations use RBPO. The result indicates that shared-policy collaboration benefits not only from graded partial-progress feedback, but also from expressing that feedback as the observed change following a revision.

The comparison separates two losses of information. Absolute pass-rate feedback remains dense, but it scores the attained state rather than the change produced by the current round; progress inherited from earlier attempts can therefore be credited repeatedly even when a later decision adds little. Final-binary feedback additionally discards partial progress and supplies only a sparse terminal signal across multiple plans, programs, and reflections. The improvement-aligned reward preserves graded execution information while aligning it with the corresponding workflow transition. This provides a possible interpretation of the observed ordering, rather than a causal explanation established by the ablation.

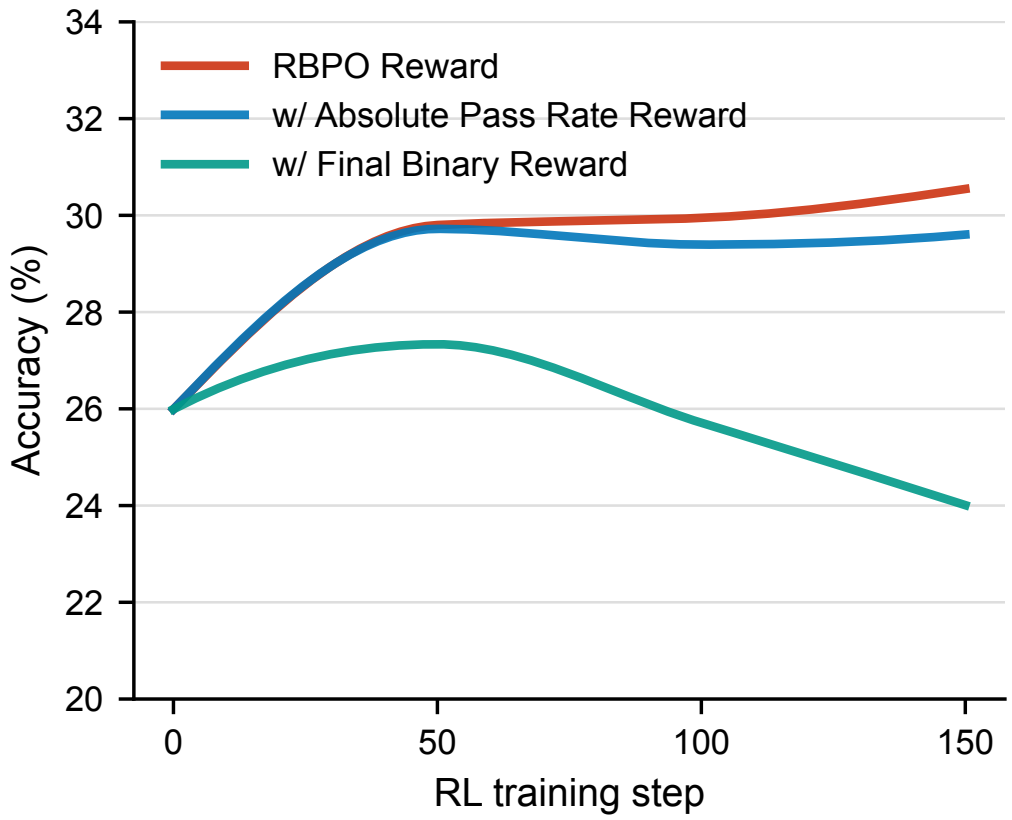


Figure S2: Reward ablation under the RBPO objective, with mean accuracy over two random seeds. From the aligned initial checkpoint, the proposed improvement-aligned reward achieves the best final accuracy and strongest late-training trajectory, the absolute pass-rate reward ranks second, and the final-binary reward performs worst.

F Ablation of the Collaborative Round Budget

We vary the maximum number of code-generation rounds $M \in \{2, 3, 4\}$ while keeping the policy, training objective, reward, and early-termination rule unchanged. A round contains a Planner response, a Coder response, and one program execution; a Reflector response is generated only when another round will be attempted.

Figure S3 shows a non-monotonic relationship between the maximum interaction budget and final accuracy. Two rounds provide the fewest opportunities to diagnose and repair a failed solution and consequently perform worst. Increasing the budget to three rounds improves performance by allowing an additional plan–code–execute–reflect cycle when the earlier attempts fail.

Extending the maximum budget from three to four rounds does not produce a further gain, although it remains stronger than the two-round configuration. This pattern suggests diminishing returns from additional interaction. Longer histories and revision drift are possible explanations, but neither is measured in this experiment. The three-round configuration therefore provides the best empirical operating point among the tested budgets rather than treating more interaction as uniformly beneficial.

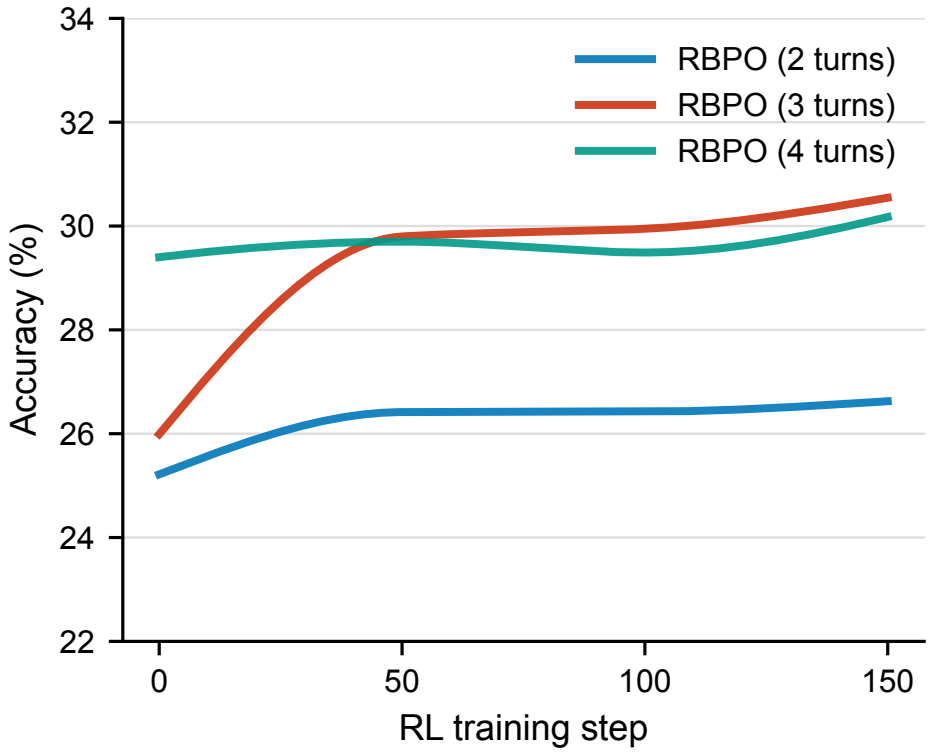


Figure S3: Ablation of the maximum collaborative round budget, with mean accuracy over two random seeds. After the initial checkpoint, three rounds achieve the strongest trajectory and best final accuracy, four rounds rank second at the final checkpoint, and two rounds perform worst. All configurations retain early termination after a successful execution.

G Summary of Supplementary Evidence

Full RBPO reaches 30.1% accuracy, followed by RBPO w/o Seq. Bal. at 29.4%, RBPO w/o Role Bal. at 27.4%, and token pooling without either level at 26.3%. The stable $39/39/22$ invocation mixture and the -11.56-point Reflector shift document the exposure and objective-mass change targeted by role balancing; RBPO w/o Seq. Bal. preserves uniform role mass but restores length dependence. At inference time, the Planner primarily improves first-attempt quality, whereas the Reflector primarily enables repair after failed execution; removing the less frequently invoked Reflector causes the larger final-accuracy drop. Incremental reward outperforms absolute pass-rate and final-binary rewards, and three rounds give the best tested performance. Together, the results support explicit and exposure-invariant represented-role weighting, functionally differentiated collaboration, and temporally aligned optimization.

H Limitations

First, we evaluate executable algorithmic problems on LiveCodeBench and DeepCoder with a fixed Planner–Coder–Reflector team. Future work should test repository-level, multi-file, long-context, and tool-use settings, together with larger, dynamic, and asynchronous teams. Second, the current invariance analysis covers within-batch actor-objective coefficients conditional on role representation. Extensions could incorporate cross-batch role presence, gradient interactions, causal credit assignment, value-objective balancing, adaptive role weights, and partial parameter sharing.

I Implementation Reference for Batch Weighting

The following reference specifies minibatch estimation of the actor objective. Let $B = |\mathcal{B}|$ be the number of responses in a complete rollout batch, and let $L_i = T_i^{-1} \sum_t \ell_{i,t}$. Sampling a minibatch $\mathcal{M}$ of m responses uniformly from this batch gives the estimator

$$\widehat{\mathcal{L}}_{\mathcal{M}} = \frac{B}{m} \sum_{i \in \mathcal{M}} \frac{\bar{\alpha}_{r_i}}{N_{r_i}} L_i. \tag{S9}$$

Each response has inclusion probability m/B, so at fixed parameters the expectation of this estimator is $\mathcal{L}_{\mathrm{RBPO}}$. Both N_r and the represented-role normalization in $\bar{\alpha}_r$ refer to the full rollout batch. Recomputing either from each microbatch generally changes the objective. With nonuniform sampling, the estimator must instead account for the actual sampling probabilities.

1. Freeze the behavior policy and collect complete collaborative trajectories, stopping on visible-test success or exhaustion of the round budget. Store response role labels, generated-token masks, and behavior-policy log probabilities.
2. Assign the current execution improvement to each plan and code response, and the next improvement to the reflection that precedes it. Compute the response advantages with the chosen base estimator.
3. Compute T_i, N_r, and $\bar{\alpha}_r$ on the complete batch. Padding and prompt tokens must be excluded from T_i and from the actor reduction. Empty or filtered responses require an explicit handling rule before these counts are formed.
4. Evaluate clipped token losses, form each L_i, and apply the batch-based response weights. If using distributed gradient averaging or accumulation, account for those additional scaling operations so that the resulting estimator has the stated normalization.
5. Optimize the actor and the base value objective, then collect new rollouts. Actor weighting alone specifies neither the critic architecture nor its loss reduction.

The estimator is unbiased for the fixed empirical actor objective at a given parameter value. Sequential optimizer steps, clipping changes, gradient accumulation, and adaptive optimizer state can make different minibatch partitions or orders produce different training trajectories. No optimizer-trajectory invariance is claimed.

Reward interpretation. For a complete trajectory with $q_{-1} = 0$, the unweighted Planner and Coder improvement rewards each telescope to $\sum_{k=0}^{K_\tau - 1} \Delta q_k = q_{K_\tau - 1}$. The Reflector rewards sum to $q_{K_\tau - 1} - q_0$. RBPO, however, reweights response losses, uses estimated advantages and PPO clipping, and may include formatting rewards. The telescoping identity therefore does not establish equivalence to maximizing final team correctness or policy-invariant reward shaping. In particular, an improved visible-test score need not imply improved performance on unseen tests.

J Scope of the Reported Compute Estimates

The aggregate compute estimates below are retained from the experimental summary. The approximation $6ND$ is motivated by parameter–token accounting for dense-model training (Hoffmann et al., 2022); its use for a collaborative RL system requires a more detailed accounting of model calls. For multiple policies indexed by j, a transparent proxy would specify

$$C_{\mathrm{update}} \approx 6 \sum_j N_j D_{j,\mathrm{update}}, \qquad C_{\mathrm{decode}} \approx 2 \sum_j N_j D_{j,\mathrm{generated}}. \tag{S10}$$

Here $D_{j,\mathrm{update}}$ counts tokens processed in actor update passes and $D_{j,\mathrm{generated}}$ counts generated tokens across all inference calls. RBPO's generated-token count must include Planner, Coder, and Reflector outputs. The update proxy is not total RL training cost: rollout generation, value/reference models, execution, and orchestration require separate accounting. Likewise, generated-token accounting omits prompt prefill, sequence-length-dependent attention costs, caching effects, and code execution at inference.

Table S3: Aggregate FLOP estimates for Qwen3-8B configurations. The training column was reported per epoch; the inference column is a generated-token proxy. The MARS^2 entry uses Qwen3-8B. The inference workload denominator remains unspecified, so these estimates alone do not establish a compute-matched accuracy comparison.

Method	Training estimate / epoch	Inference estimate
GRPO	1.22×10^{20}	1.024×10^{16}
MARS^2 (Qwen3-8B)	2.44×10^{20}	2.051×10^{16}
RBPO	1.22×10^{20}	1.024×10^{16}

The reported ratios are 50.0% lower training and 50.1% lower inference estimates for RBPO relative to the listed MARS^2 entry. These are ratios of the supplied proxy values, not measured wall-clock or end-to-end savings. Sharing policy weights can reduce distinct parameter storage, but does not automatically reduce FLOPs for an unchanged total number of processed tokens. A reproducible efficiency comparison needs model sizes and token counts for every role, prompt and generation lengths, search/rollout budgets, the number of evaluated problems, and measured hardware time under a common workload.

K Workflow-Induced Role Exposure

This analysis concerns complete trajectories under the rollout definition in Section 3, before any response-level filtering or resampling. For $J \geq 1$ trajectories, let $K_\tau \in \{1, \ldots, M\}$ be the number of executed programs, $L = \sum_\tau K_\tau$, and $\bar{K} = L/J$. Every trajectory has one plan and one code response for each executed program, and one reflection between consecutive programs. Thus,

$$N_{\mathrm{P}} = N_{\mathrm{C}} = L, \qquad N_{\mathrm{R}} = L - J, \qquad B = 3L - J. \tag{S11}$$

The response-pooled role masses are

$$Q_{\mathrm{resp}}(\mathrm{P}) = Q_{\mathrm{resp}}(\mathrm{C}) = \frac{\bar{K}}{3\bar{K} - 1}, \qquad Q_{\mathrm{resp}}(\mathrm{R}) = \frac{\bar{K} - 1}{3\bar{K} - 1}. \tag{S12}$$

Since $1 \leq \bar{K} \leq M$ and

$$\frac{\mathrm{d}}{\mathrm{d}\bar{K}} \frac{\bar{K} - 1}{3\bar{K} - 1} = \frac{2}{(3\bar{K} - 1)^2} > 0, \tag{S13}$$

the Reflector mass lies between zero and $(M - 1)/(3M - 1)$. For three rounds, it cannot exceed $1/4$, even when all trajectories use the full budget. Whenever $\bar{K} > 1$, reflection is represented, yet its response-pooled mass is below the uniform three-role target by

$$\frac{1}{3} - Q_{\mathrm{resp}}(\mathrm{R}) = \frac{2}{3(3\bar{K} - 1)}. \tag{S14}$$

At $\bar{K} = 1$, reflection is absent; RBPO then assigns one half to each of the two represented roles rather than assigning mass to nonexistent responses.

The identity isolates a structural source of imbalance without assumptions about rewards or model quality. If earlier success reduces $\bar{K}$, response pooling reduces the Reflector coefficient further. This is a conditional implication, not a claim that such a trend was measured during training. Token pooling introduces an additional dependence on output lengths and need not satisfy the same $1/4$ bound. The empirical collected-response audit in Appendix C is reported separately; the identities here do not reconstruct its counts or resolve response-filtering conventions.

L Factorial Reading of the Four Objective Results

Let A_{sr} be the reported mean accuracy, with $s \in \{0, 1\}$ indicating sequence balancing and $r \in \{0, 1\}$ role balancing. Table 2 provides all four entries:

$$A_{00} = 26.3, \quad A_{10} = 27.4, \quad A_{01} = 29.4, \quad A_{11} = 30.1. \tag{S15}$$

The conditional role-balancing gains are $A_{01} - A_{00} = 3.1$ and $A_{11} - A_{10} = 2.7$ percentage points. The conditional sequence-balancing gains are $A_{10} - A_{00} = 1.1$ and $A_{11} - A_{01} = 0.7$ points. Both components therefore have positive contrasts under either setting of the other in the reported means.

Averaging the two contrasts for each component gives descriptive factorial effects of 2.9 points for role balancing and 0.9 points for sequence balancing. Their sum is the 3.8-point difference between full RBPO and token pooling. This is a symmetric arithmetic decomposition of the four means, not an allocation of causal credit within a trained model. The interaction contrast is

$$A_{11} - A_{10} - A_{01} + A_{00} = -0.4 \text{ percentage points.} \tag{S16}$$

Accordingly, the data support benefits of both components but do not support a claim of super-additive synergy. Without per-run uncertainty estimates, these contrasts cannot establish statistical significance or consistent ordering in every seed. They require no additional experimental measurements and preserve the reported mean-only presentation.