# COUNTERFACTUAL CREDIT ASSIGNMENT OVER EXECUTION GRAPHS OF LLM-BASED MULTI-AGENT SYSTEMS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Training LLM-based multi-agent systems (MAS) requires assigning credit to agents for how their actions influence subsequent interactions and outcomes. We present ROLLCREDITS, a method for assigning decision-level credit through counterfactual advantage estimation. At a shared history, the expected reward difference between an action and policy-sampled alternatives equals the acting agent's advantage. We formalize these comparisons using execution graphs to represent agent interactions, including sequential, parallel, and multi-round structures, with potentially distinct rewards for different agents. Each candidate action completes the downstream execution it induces and is scored by the acting agent's reward, and we establish sampling and replay conditions under which the resulting group-relative estimator is unbiased after a fixed group-size correction. To keep collection affordable, candidates branch from decision states saved during seed executions and their continuations run concurrently. Across mathematical reasoning, code generation, and competitive games, ROLLCREDITS leads in four code metrics, three math metrics, and three games, including a held-out game. Diagnostics against an independently sampled reference show nearly unbiased credit, while local feedback retains bias despite additional sampling. In controlled math and game experiments, saved-state reuse reduces prompt recomputation by 24–25% compared with ancestor replay.

## 1 INTRODUCTION

LLM-based multi-agent systems (MAS) can improve the ability to solve complex problems through division of labor and interaction among agents (Hong et al., 2023; Qian et al., 2025; Hadfield et al., 2025). Reinforcement learning offers a way to improve model behavior using reward feedback (Ouyang et al., 2022; Shao et al., 2024; Guo et al., 2025), and recent frameworks apply it to learning multi-agent interactions (Zhang et al., 2026; Zhao et al., 2025). Evaluating completed outcomes provides a practical source of training feedback (Cobbe et al., 2021; Le et al., 2022). The final outcome depends on actions taken by different agents and the interactions that follow. Assigning credit to these actions requires accounting for their downstream effects. This raises the problem of accurate and efficient credit assignment when training MAS from outcome rewards.

An execution's outcome reward reflects the combined effects of several agents' actions, leaving their respective contributions unclear (Foerster et al., 2018). An action may improve the chance of success even when later actions lead to failure. Comparing complete executions also mixes variation in the decision history with variation in the action and its continuation (Figure 1a). Existing methods obtain more local comparisons by grouping actions at repeated states or within the same agent's turn (Feng et al., 2025; Zhao et al., 2025), while agent-specific normalization calibrates training signal scales (Feng et al., 2026). Intermediate action quality can be estimated through learned critics (Foerster et al., 2018) or process reward models, including models trained with automatically generated step evaluations (Wang et al., 2024), providing local action evaluations (Figure 1b). Such evaluations judge an action without the responses it induces: in solver–verifier interaction, a nearly correct answer can lead the verifier to identify the missing step and the solver to revise correctly, whereas a fundamentally wrong answer can mislead the verifier as well, yet both answers are simply incorrect in isolation. Direct rollout methods instead estimate intermediate values from sampled



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.

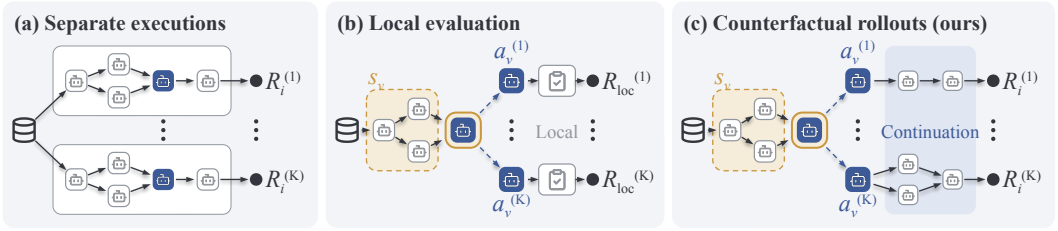


Figure 1: Credit signals for a decision in MAS. (a) Separate executions mix differences in history and action. (b) Local evaluation scores candidates $a_v^{(1)}, \ldots, a_v^{(K)}$ at a shared history $s_v$ (shaded), without their downstream responses. (c) RollCredits completes each candidate's continuation and compares the acting agent's outcome rewards. Filled tiles mark the acting agent; the outlined call marks the selected decision $v$.

continuations, using repeated rollouts or rollout trees (Kazemnejad et al., 2025; Wang et al., 2026). These methods provide more fine-grained training feedback. A central challenge is to make this feedback accurately reflect the expected reward differences between alternative actions, while controlling the cost of evaluating their subsequent interactions.

Our starting point is the counterfactual interpretation of advantages in multi-agent credit assignment (Foerster et al., 2018). At the same history, the expected reward difference between an action and policy-sampled alternatives equals the acting agent's advantage. Executing each alternative to completion therefore provides samples for estimating credit, including the effects of subsequent responses. MAS execution can involve branching dependencies (Zhuge et al., 2024; Qian et al., 2025) and potentially distinct agent rewards, consistent with the need to accommodate heterogeneous preferences in LLM applications (Sun et al., 2026). We introduce RollCredits, a principled approach to credit assignment under these execution and reward conditions. We use a unified graph representation of finite MAS executions, covering sequential, parallel, and multi-round interactions. This representation specifies the history shared by candidate actions, each of which completes the downstream execution it induces under the collection policy (Figure 1c). Comparing their outcomes using the acting agent's reward yields a group-relative advantage estimate, with sufficient conditional sampling assumptions for unbiased estimation after a fixed group-size correction.

To reduce collection cost, RollCredits launches concurrent candidate continuations from decision states saved during seed executions, avoiding repeated execution of shared ancestors. We evaluate RollCredits across MAS topologies for mathematical reasoning, code generation, and competitive games. Among the evaluated methods, it achieves the highest aggregate MAS pass@8 in math and code and the highest win rate in three of four games, including a held-out game. Further analyses assess credit quality and computation reuse during collection. Our contributions are:

1. A unified graph formulation of decision-level credit across sequential, parallel, and multi-round MAS executions with potentially distinct agent rewards (§2).

2. A counterfactual rollout estimator that evaluates actions through their complete continuations, with sampling and replay conditions under which the advantage estimate is unbiased (§3).

3. A seed-based collection procedure that branches candidates from saved decision states, reusing the computation of their shared history, and runs their continuations concurrently (§3.3).

4. An evaluation on mathematical reasoning, code generation, and competitive games, with direct measurements of credit bias and variance and of computation reuse during rollout collection (§4).

## 2 PROBLEM FORMULATION

We formalize LLM-based multi-agent executions as graphs with potentially distinct agent rewards. At each decision, we study the acting agent's advantage relative to alternatives sampled at the same history, with subsequent actions following a fixed collection policy.

**Execution graphs.** To extend decision-level credit assignment beyond sequential interactions, we represent MAS execution as a directed acyclic graph (DAG) (Zhuge et al., 2024; Qian et al., 2025). For a finite realized run, let $G = (V, E)$ denote its execution graph, whose edges encode dependencies between model calls. Each node $v \in V$ is a single model call. The map $\iota : V \to \{1, \dots, n\}$ assigns each call to one of the $n$ agents, so $\iota(v)$ is the index of the agent acting at $v$. All agents share a policy $\pi_\theta$ with parameters $\theta$; $\mu$ denotes the version used to collect rollouts. Parameter sharing aims to consolidate capabilities learned through different agent roles within a single foundation model. The action $a_v$ is a complete text turn sampled from $\mu(\cdot \mid s_v)$, where the decision history

$$s_v \;=\; \big(x,\ \{(u, a_u) : u \in \mathrm{anc}(v)\},\ o_v\big) \tag{1}$$

collects the task input $x$, the actions of all ancestor nodes in $\mathrm{anc}(v)$, and $o_v$, which records the acting agent's role instructions and observations available before call $v$. The policy accesses only the information in $s_v$ visible to that role. This representation includes chains, fan-out to parallel workers, and fan-in to an aggregator; multi-round interactions are unrolled into distinct nodes. The execution graph may develop dynamically as agents' actions determine which model calls occur next. An alternative action may therefore induce a different downstream graph. In a chain, the ancestor actions form a transcript prefix; in a general DAG, they follow the dependencies of $v$.

**Rewards and advantages.** Each agent $i$ receives an outcome reward $R_i$ at the end of execution. We assume access to outcome rewards for completed executions, without requiring decision-level reward annotations or a separately trained process reward model. We allow arbitrary relationships among agent rewards. For example, cooperative agents share $R_i \equiv R_{\mathrm{team}}$, whereas two-agent zero-sum settings satisfy $R_1 = -R_2$. General-sum settings impose neither constraint.

At node $v$, let $i = \iota(v)$ denote the acting agent and $\mu$ the policy used to collect rollouts. We evaluate action $a$ by agent $i$'s expected outcome reward given the history $s_v$, with all subsequent actions sampled from $\mu$. Let $\tau$ denote a completed execution and $P^\mu(\cdot \mid s_v, a)$ its conditional distribution. The action value, baseline, and advantage are

$$\begin{aligned} Q_i^\mu(s_v, a) &= \mathbb{E}_{\tau \sim P^\mu(\cdot \mid s_v, a)}\big[R_i\big], \\ b_i^\mu(s_v) &= \mathbb{E}_{a' \sim \mu(\cdot \mid s_v)}\big[Q_i^\mu(s_v, a')\big], \\ A_i^\mu(s_v, a) &= Q_i^\mu(s_v, a) - b_i^\mu(s_v). \end{aligned} \tag{2}$$

The baseline $b_i^\mu(s_v)$ is the expected reward under actions sampled from $\mu$ at the same history. The advantage $A_i^\mu(s_v, a)$ measures how much action $a$ improves the acting agent's expected reward over this baseline. Under a shared reward, it measures the corresponding improvement in expected team reward.

## 3 METHOD

ROLLCREDITS trains a shared MAS policy using counterfactual advantage estimates from agent-specific outcome rewards. It reuses decision states from seed executions and schedules candidate continuations concurrently to reduce the cost of collecting these estimates (Figure 2).

### 3.1 OPENING SCENE: A COUNTERFACTUAL VIEW OF ADVANTAGES

We begin with the counterfactual comparison implicit in the advantage definition in equation 2. For agent $i = \iota(v)$, compare the expected reward $Q_i^\mu(s_v, a)$ with that of an alternative action $a'$ at the same history. Subsequent actions follow $\mu$ conditioned on the history induced by the chosen action. Averaging over policy-sampled alternatives gives

$$\begin{aligned} &\mathbb{E}_{a' \sim \mu(\cdot \mid s_v)}\left[Q_i^\mu(s_v, a) - Q_i^\mu(s_v, a')\right] \\ &\qquad = Q_i^\mu(s_v, a) - b_i^\mu(s_v) = A_i^\mu(s_v, a). \end{aligned} \tag{3}$$

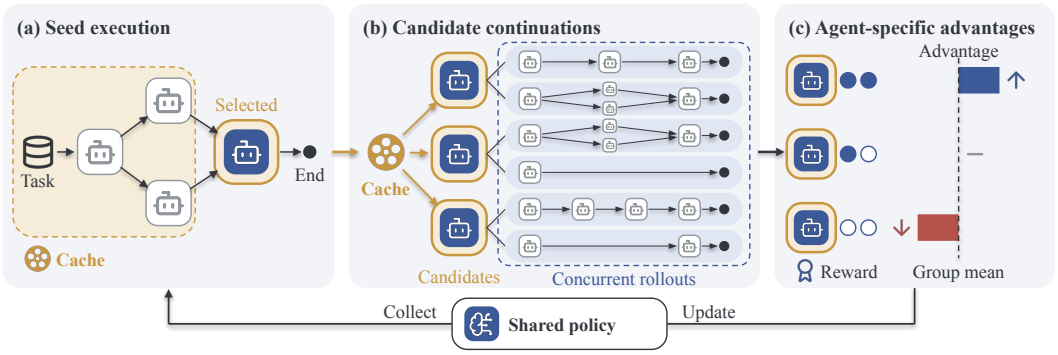


Figure 2: Training loop of ROLLCREDITS. (a) A seed execution saves a decision state, while the inference backend caches the shared prompt prefix. (b) Each of $K$ candidate actions receives $M$ concurrent continuations, scored by the acting agent's outcome reward. (c) Candidate mean rewards are centered within the group to form raw group-relative estimates, then standardized for updating the shared policy.

Thus, the expected counterfactual reward difference is the advantage itself.

This identity determines how to construct the comparison: preserve the graph-defined history $s_v$, sample alternatives from $\mu(\cdot \mid s_v)$, and evaluate every outcome using the acting agent's reward $R_i$. We identify each comparison group by $(x, v, s_v)$ and sample $K \geq 2$ candidate actions $a_1, \ldots, a_K$, writing $a_k \equiv a_v^{(k)}$ for the $k$th sampled output of the same call $v$. Using $R_i$ keeps the comparison aligned with that agent's advantage even when other agents execute subsequent calls.

To estimate each candidate's value, a completed continuation sampled from $P^\mu(\cdot \mid s_v, a_k)$ supplies a reward with conditional expectation $Q_i^\mu(s_v, a_k)$. We therefore execute candidates to completion under $\mu$, obtaining value estimates directly from outcome rewards. For a fixed topology, this requires re-executing the candidate's downstream nodes. When actions determine subsequent calls, each candidate follows the downstream graph it induces. Although the collection policy is fixed, subsequent agents respond to each candidate's induced history. The comparison therefore captures its effects on later decisions.

### 3.2 IN FOCUS: AGENT-SPECIFIC ADVANTAGE ESTIMATION

Let $R_i^{(k,m)}$ be the acting agent $i = \iota(v)$'s reward from continuation $m$ of candidate $a_k$. Averaging over $M \geq 1$ continuations gives

$$\widehat{Q}_i(s_v, a_k) = \frac{1}{M} \sum_{m=1}^{M} R_i^{(k,m)}. \tag{4}$$

We then center each estimate by the mean of its comparison group:

$$\widehat{b}_i^{\text{group}} = \frac{1}{K} \sum_{j=1}^{K} \widehat{Q}_i(s_v, a_j), \qquad \widehat{A}_{i,k} = \widehat{Q}_i(s_v, a_k) - \widehat{b}_i^{\text{group}}. \tag{5}$$

This is the raw group-relative estimate before correction or standardization. The following assumption connects the sampled rewards to the history-conditioned values defining the advantage.

**Assumption 1** (Conditional branch sampling)**.** *Fix a node $v$ with history $s_v$. (i) The candidates are drawn i.i.d. from $\mu(\cdot \mid s_v)$. (ii) Conditional on $s_v$ and the candidate actions, the continuations are independent, and each continuation of candidate $k$ has distribution $P^\mu(\cdot \mid s_v, a_k)$. (iii) $M$ is fixed before any candidate action or reward is observed, and outcome rewards have finite expectations.*

**Proposition 1** (Unbiased agent-specific advantage estimation)**.** *Under Assumption 1, for every node $v$ with $i = \iota(v)$ and every candidate $k$, the group-relative estimator satisfies*

$$\mathbb{E}\left[\frac{K}{K-1} \widehat{A}_{i,k} \mid s_v, a_k\right] = Q_i^\mu(s_v, a_k) - b_i^\mu(s_v) = A_i^\mu(s_v, a_k).$$

**Algorithm 1** ROLLCREDITS training with counterfactual rollouts

**Require:** Shared policy $\pi_\theta$; candidate count $K \geq 2$; continuation count $M \geq 1$

**for** each training iteration **do**
  Fix $\mu \leftarrow \pi_\theta$; initialize training batch $\mathcal{B} \leftarrow \varnothing$
  Generate seed executions under $\mu$ and save selected decision states
  **for** each saved state at node $v$ with history $s_v$ **do**
    Set $i \leftarrow \iota(v)$; sample $a_1, \ldots, a_K$ i.i.d. from $\mu(\cdot \mid s_v)$
    Record the sampling log-probability of each candidate token
    For each $a_k$, execute $M$ complete continuations under $\mu$ using the replay conditions in Appendix C.1
    Score every outcome with $R_i$ and estimate $\widehat{Q}_i(s_v, a_k)$ by equation 4
    Compute the raw group-relative estimates $\widehat{A}_{i,k}$ by equation 5
    If all candidate value estimates are equal, **continue**
    Standardize advantages within the group
    Add candidate actions and contexts, token log-probabilities, and advantages to $\mathcal{B}$
  **end for**
  Update $\theta$ on nonempty $\mathcal{B}$ by maximizing equation 6, with credit on candidate tokens
**end for**

*The factor $K/(K-1)$ corrects for including the evaluated candidate in the group mean. The result holds for arbitrary agent rewards $\{R_i\}$, with subsequent actions sampled from $\mu$.*

Appendix A gives the proof. Proposition 2 in Appendix C.1 connects graph replay to the conditional reward means used in this argument, including when candidates share saved state beyond $s_v$.

### 3.3 ROLLING THE CREDITS: COLLECTION AND OPTIMIZATION

A seed execution is a rollout under $\mu$ used to discover decision states for counterfactual comparisons. We save selected states and launch candidate continuations from them, reusing shared ancestor computation. Downstream replay regenerates outputs affected by the candidate action and follows the calls and termination decisions it induces. Reuse of additional saved information follows the conditions in Appendix C.1. Branches run concurrently in separate environment copies under the same collection policy $\mu$. Algorithm 1 gives the training procedure; scheduling details are provided in Appendix C.3.

Training uses $\widehat{A}_{i,k}$ without the fixed group-size correction, preserving signs and candidate ordering for a fixed $K$. We discard groups with identical candidate value estimates, standardize the remaining groups' advantages, and assign each candidate's advantage to all its tokens. Continuations evaluate the focal candidate; downstream actions receive their own credit when selected for separate comparisons. Let $t$ index a retained token, $a_t$ its value, and $s_t$ its input context, including the preceding tokens of the action. With $\bar{\mu}_t$ denoting the policy that generated the token and $\widehat{A}_t$ its standardized training advantage, we maximize the clipped objective

$$\mathcal{L}(\theta) = \mathbb{E}_t\Big[\min\big(\rho_t \widehat{A}_t,\ \mathrm{clip}(\rho_t, 1-\epsilon_l, 1+\epsilon_h)\widehat{A}_t\big)\Big], \qquad \rho_t = \frac{\pi_\theta(a_t \mid s_t)}{\bar{\mu}_t(a_t \mid s_t)}. \tag{6}$$

The expectation is over retained training tokens, and $\epsilon_l$ and $\epsilon_h$ are the lower and upper clipping thresholds. Standardization and filtering change the weighting of the updates; Proposition 1 concerns the raw estimator before these operations. Their relation to the group-size correction and shared-parameter updates is discussed in Appendix C.2.

## 4 EXPERIMENTS

We evaluate ROLLCREDITS on mathematical reasoning, code generation, and competitive games, covering different MAS execution topologies and both shared and agent-specific rewards. We examine task performance, counterfactual credit quality, and collection efficiency.

Table 1: Performance across three domains (%). Math and code report aggregate pass@$k$ under single-agent and MAS inference. Games report seat-balanced win rates against frozen Qwen3-4B, with draws included in the denominator. C4, Dice, and BT denote Connect Four, Liar's Dice, and held-out (out-of-distribution, OOD) Breakthrough. Base is the initial Qwen3-4B checkpoint before our RL training; it is the games opponent and is therefore not evaluated there. Bold marks the highest point estimate among the trained methods in each column.

| | **Math: pass@$k$** | | | | **Code: pass@$k$** | | | | **Games: win rate** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Single | | MAS | | Single | | MAS | | Training | | | OOD |
| Method | pass@1 | pass@8 | pass@1 | pass@8 | pass@1 | pass@8 | pass@1 | pass@8 | Blotto | C4 | Dice | BT |
| Base | 47.2 | 66.4 | 48.4 | 68.1 | 46.8 | 56.2 | 50.1 | 62.3 | – | – | – | – |
| GRPO | 65.3 | 79.2 | 65.4 | 77.8 | 55.8 | 68.7 | 57.6 | 70.8 | 16.6 | 56.8 | 54.1 | 24.6 |
| traj-GRPO | 57.0 | 69.6 | 71.3 | 85.2 | 49.0 | 61.4 | 53.7 | 65.9 | 26.0 | 53.7 | 23.1 | 58.6 |
| Dr. MAS | 63.1 | 77.9 | **73.2** | 84.5 | 49.7 | 62.8 | 53.6 | 66.5 | 93.8 | 62.5 | 61.1 | 70.3 |
| AT-GRPO | 64.6 | 80.8 | 67.2 | 82.8 | 49.8 | 62.3 | 55.2 | 68.4 | 67.4 | **65.8** | 60.9 | 55.3 |
| ROLLCREDITS | **66.5** | **81.0** | 71.1 | **86.0** | **56.1** | **69.2** | **60.1** | **73.0** | **99.0** | 60.4 | **75.8** | **84.8** |

### 4.1 EXPERIMENTAL SETUP

**Tasks and policies.** Math uses solver–verifier interactions, in which the solver revises its answer after a rejection; code uses planner–coder–tester interactions, in which the coder revises its program after testing; and games involve competing agents acting in a shared environment. Math and code assign shared outcome rewards, while games assign agent-specific zero-sum rewards. All policies start from Qwen3-4B (Yang et al., 2025), with parameters shared across roles. Task-specific implementation details and execution graphs are provided in Appendix B.

**Baselines.** Our baselines include GRPO with single-agent training (Shao et al., 2024); trajectory-level GRPO (traj-GRPO), which computes relative advantages across complete MAS executions; Dr. MAS, which calibrates advantages using agent-specific reward statistics (Feng et al., 2026); and AT-GRPO, which compares candidate actions at a shared state for each agent and turn (Zhao et al., 2025).

**Evaluation.** All checkpoints follow a common evaluation protocol within each domain. Math and code use eight samples per problem under both single-agent and MAS inference. Game checkpoints are evaluated in both seats against a frozen Qwen3-4B opponent. Evaluation protocols are detailed in Appendix B.3; complete results and bootstrap confidence intervals appear in Appendix B.4.

### 4.2 TASK PERFORMANCE

Table 1 reports aggregate results. Pass@1 is the mean correctness of the eight samples per problem, and pass@8 is the fraction of problems solved by at least one sample. Math aggregates weight benchmarks equally, while code aggregates weight problems equally. ROLLCREDITS improves over the initial Qwen3-4B checkpoint in math and code and achieves the highest win rate among the evaluated methods in three of four games.

**Mathematical reasoning.** ROLLCREDITS achieves the highest single-agent scores in both metrics, with macro-average pass@1 of 66.5% and pass@8 of 81.0%, and also leads in MAS pass@8 at 86.0%. All MAS-trained methods benefit from MAS inference, with pass@1 rising from 66.5% to 71.1% for ROLLCREDITS and from 63.1% to 73.2% for Dr. MAS. GRPO gains little under this protocol, moving from 65.3% to 65.4%. These results show that ROLLCREDITS retains strong single-agent performance while benefiting from verification and revision under MAS inference. Additional revisions at inference also improve its math pass@1 (Appendix B.5). Its credit estimates evaluate solver actions through their downstream outcomes; Section 4.3 examines this mechanism directly.

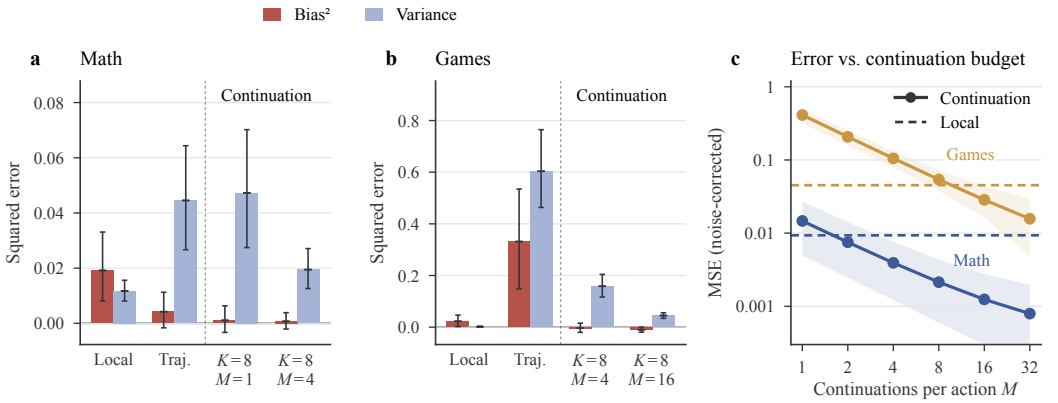


Figure 3: Credit-estimation error. (a,b) Squared bias and variance relative to the policy-baseline advantage on 48 math histories and 24 late-turn game states; continuation estimates use $K{=}8$, with $M{=}1, 4$ in math and $M{=}4, 16$ in games, on matched histories and candidate actions within each domain. (c) MSE versus $M$ for advantages centered over four fixed candidates, on 24 math and 24 game states: continuation estimates (solid) and local feedback (dashed). Noise-corrected estimates can be slightly negative. Bars and shading show 95% bootstrap intervals over histories.

**Code generation.** ROLLCREDITS achieves the highest point estimate in all four code columns. Under MAS inference, its pass@1 and pass@8 reach 60.1% and 73.0%, compared with 57.6% and 70.8% for GRPO, the strongest baseline in both metrics; under single-agent inference, it also leads at 56.1% and 69.2%. As in math, ROLLCREDITS gains more from MAS inference than GRPO, 4.0 versus 1.8 points in pass@1, while retaining the strongest single-agent coder.

**Competitive games.** ROLLCREDITS achieves the highest win rate in three of the four games. It wins 99.0% of games on Colonel Blotto and 75.8% on Liar's Dice, compared with 93.8% and 61.1% for Dr. MAS, the strongest baseline in both, and 84.8% on held-out Breakthrough, against 70.3%. In all three, the confidence intervals of ROLLCREDITS and the strongest baseline do not overlap. ROLLCREDITS evaluates alternative moves through complete continuations, accounting for the opponent's responses and subsequent play. The gain on held-out Breakthrough suggests that the learned policy generalizes beyond the training games.

### 4.3 COUNTERFACTUAL CREDIT QUALITY

Credit quality concerns how closely the signal assigned to an action tracks its advantage $A_i^{\mu}(s_v, a)$ at a fixed history. We freeze the reported checkpoints and compare credit sources against an independently sampled reference. We refer to counterfactual advantage estimates from complete continuations as continuation estimates.

**Diagnostic protocol.** We compare local feedback, trajectory-level credit, and continuation estimates at $K{=}8$ on 48 math histories ($M{=}1, 4$) and 24 game states at predetermined late turns ($M{=}4, 16$). At each history, two sampled actions are held fixed across eight repeated comparison groups. Independent samples estimate each action's value and the policy baseline, using 32 complete continuations for each. We decompose advantage-estimation error into squared bias and variance, correcting for finite-repeat and reference noise. A separate fixed-candidate analysis examines how error changes with the continuation budget. Appendix B.6 gives the source definitions and sampling design.

**Bias of credit sources.** Local feedback evaluates an intermediate action rather than its downstream outcome. Its squared bias is 0.019 in math and 0.023 in games, with both 95% intervals excluding zero. In math, verification can repair an incorrect answer or lead to revision of a correct one; in games, local feedback is usually zero before termination. Trajectory-level credit also uses a baseline from independent executions rather than alternatives at the same history, and its squared bias reaches

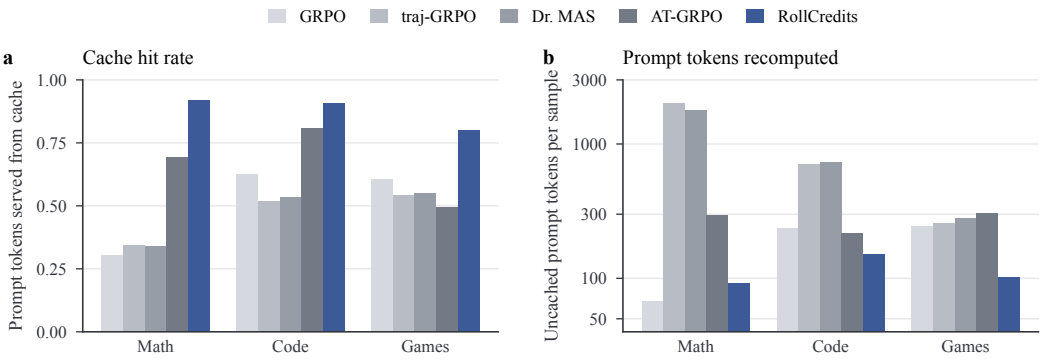


Figure 4: Prefix-cache reuse during training collection, averaged over all training updates of the checkpoints in Table 1. (a) Fraction of prompt tokens in each training sample that the inference backend serves from its prefix cache. (b) Prompt tokens per training sample that are not served from the cache and must be recomputed, on a logarithmic scale. Counts cover the samples used for policy updates.

0.331 at late-turn game states. Continuation estimates instead have noise-corrected squared bias near zero in both domains (Figure 3a,b), consistent with Proposition 1.

**Variance and sampling budget.** At fixed $K{=}8$, increasing $M$ from 1 to 4 reduces continuation-estimate variance from 0.047 to 0.019 in math. Games exhibit greater variance, motivating a follow-up at $M{=}16$: variance falls from 0.158 at $M{=}4$ to 0.045. At $M{=}4$, continuation estimates have the lowest MSE point estimate in math: 0.020, compared with 0.031 for local feedback and 0.049 for trajectory-level credit. In games, increasing $M$ from 4 to 16 reduces MSE from 0.154 to 0.035, approaching local feedback's 0.024 while remaining well below trajectory-level credit's 0.935. The fixed-candidate analysis isolates this variance: averaging $M$ continuations gives an error curve of $\beta^2 + \sigma^2/M$, which falls below local feedback at approximately $M{=}1.6$ in math and $M{=}10$ in games (Figure 3c). These curves center advantages over fixed candidates, whereas panels (a,b) use the policy baseline with resampled comparison actions. Complete continuations thus provide nearly unbiased credit whose precision improves with sampling; averaging more samples from an unchanged local feedback estimator does not remove its bias.

### 4.4 Collection efficiency

We measure prefix-cache reuse during training and the reduction in prompt recomputation from restoring saved decision states. All methods generate with SGLang (Zheng et al., 2024), whose radix-tree prefix cache reuses computation of previously processed prompt prefixes.

**Cached shared histories.** Candidates at a saved decision state share the complete history up to that state, and only their own actions and continuations differ. The inference backend serves a large fraction of prompt tokens in training samples from its prefix cache: 92% in math, 91% in code, and 80% in games, compared with at most 69%, 81%, and 61% for any baseline (Figure 4a). Although the shared history makes the context of RollCredits long, averaging 1,522 prompt tokens per training sample in math, only 92 of them need to be recomputed. In code and games, RollCredits recomputes the fewest prompt tokens per sample of all methods, 151 and 102, against 218–739 and 246–304 for the baselines (Figure 4b). In math and code, traj-GRPO and Dr. MAS recompute the most, since they train on independent executions that share only the task prompt. Comparing actions at a shared history thus not only isolates their credit (Section 4.3) but also lets the collector reuse the computation of that history.

**Saved decision states.** Without saved states, a collector must rebuild the shared MAS history before each candidate can act. In a controlled measurement at 32 fixed decision states with the final checkpoint (Appendix B.7), starting candidates from the saved states reduces recomputed prompt

tokens by 24% in math and 25% in games relative to ancestor replay, which repeats the ancestor turns for every candidate, even though both collectors use the same prefix cache. Prefix caching reuses computation within inference requests, while saved-state reuse avoids issuing ancestor requests to reconstruct the history. The controlled comparison measures this additional benefit with caching enabled.

## 5 Related Work

**Reinforcement learning for MAS.** Reinforcement learning for MAS spans cooperative task solving (Liu et al., 2025b) and competitive self-play in zero-sum games (Liu et al., 2025a). MARFT develops a fine-tuning framework for heterogeneous agent architectures and asynchronous interactions (Liao et al., 2025), while MARTI integrates multi-agent workflows with distributed policy training and verifiable or generative rewards (Zhang et al., 2026). Hierarchical approaches also train planners and tool-executing sub-agents as distinct models (Hong et al., 2025). Graph-based frameworks such as GPTSwarm and MacNet organize agent interactions through explicit dependencies (Zhuge et al., 2024; Qian et al., 2025). Our setting trains a shared policy across roles with potentially distinct outcome rewards.

**Credit assignment in MAS.** AT-GRPO groups candidate actions by agent and turn at a shared state (Zhao et al., 2025), while Dr. MAS calibrates advantage scales using agent-specific reward statistics (Feng et al., 2026). Other approaches use process reward models (Lightman et al., 2023), counterfactual critics (Foerster et al., 2018), or team-value factorization (Sunehag et al., 2017; Rashid et al., 2018) to construct training signals. Recent studies further explore reward decomposition for sequential cooperative teams (Deshmukh et al., 2026) and complete counterfactual rollouts from a shared decision history (Chen et al., 2026). Our method estimates the acting agent's advantage from outcome rewards, with graph-defined comparison histories and replay conditions for candidate-induced continuations.

**Efficient rollout collection.** Complete continuations add generation cost, motivating state reuse and concurrent scheduling. Tree-based sampling reduces repeated prefix computation through branching at agent interaction steps (Ji et al., 2025) or within generated sequences (Li et al., 2025). For MAS, dependency-aware parallel sampling and hierarchical load balancing improve rollout efficiency (Jiang et al., 2026). Our collection reuses states from seed executions to avoid repeating shared ancestor computation and schedules candidate branches concurrently in separate environment copies. The execution dependencies specify which saved outputs can be retained and which calls must be regenerated, subject to the replay conditions in Appendix C.1.

## 6 Conclusion

We presented RollCredits, which trains MAS from outcome rewards by estimating action advantages through complete continuations. A unified graph formulation specifies the shared history and the acting agent's reward for each comparison. Under the stated sampling conditions, the group-relative estimate is unbiased after a fixed group-size correction. Our diagnostics find nearly unbiased estimates, with variance controlled by the sampling budget, while averaging more samples from an unchanged local feedback estimator does not remove its bias. Reusing saved decision states reduces prompt recomputation from ancestor replay. Experiments show strong performance across mathematical reasoning, code generation, and competitive games, including a held-out game. These results support credit assignment based on actions' downstream consequences as a practical approach to training LLM-based multi-agent systems.

## 7 Limitations

Our evaluation covers three MAS execution topologies spanning mathematical reasoning, code generation, and games; resource constraints leave other topologies unexplored. Completing each counterfactual branch can be costly in MAS with long interactions or expensive environment calls. Future work could explore conditions for joint improvement and equilibrium convergence under shared-parameter optimization.

### AI USE STATEMENT

In this work, we used generative AI tools for language polishing of the manuscript and code implementation. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

### REPRODUCIBILITY STATEMENT

The assumptions behind Proposition 1 are stated in Section 3, with proofs and replay conditions in Appendices A and C.1. Appendix B describes the task environments, training hyperparameters, evaluation protocols, and diagnostic procedures, and Appendix D gives the full prompts of every agent. All experiments start from a public base model and use publicly available training and evaluation data. An anonymized version of the code is provided in the supplementary material. We will release the source code publicly upon publication.

## REFERENCES

Yanjun Chen, Yirong Sun, Hanlin Wang, Jinghan Wang, Xinming Zhang, Xiaoyu Shen, Wenjie Li, and Wei Zhang. Exact is easier: Credit assignment for cooperative llm agents, 2026. URL https://arxiv.org/abs/2603.06859.

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

Shripad Deshmukh, Jayakumar Subramanian, Raghavendra Addanki, and Nikos Vlassis. Cosac: Counterfactual credit assignment in sequential cooperative teams, 2026. URL https://arxiv.org/abs/2604.17693.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. In *Neural Information Processing Systems*, 2025. doi: 10.48550/arXiv.2505.10978. URL https://arxiv.org/abs/2505.10978. arXiv:2505.10978.

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.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence*, 2018. doi: 10.1609/aaai.v32i1.11794. URL https://arxiv.org/abs/1705.08926. arXiv:1705.08926.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou,

Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z. URL https://doi.org/10.1038/s41586-025-09422-z.

Jeremy Hadfield, Barry Zhang, Kenneth Lien, Florian Scholz, Jeremy Fox, and Daniel Ford. How we built our multi-agent research system. Anthropic Engineering, June 2025. URL https://www.anthropic.com/engineering/multi-agent-research-system. Published June 13, 2025.

Haoyang Hong, Jiajun Yin, Yuan Wang, Jingnan Liu, Zhe Chen, Ailing Yu, Ji Li, Zhiling Ye, Hansong Xiao, Yefei Chen, Hualei Zhou, Yun Yue, Minghui Yang, Chunxiao Guo, Junwei Liu, Peng Wei, and Jinjie Gu. Multi-agent deep research: Training multi-agent systems with m-grpo, 2025. URL https://arxiv.org/abs/2511.13288.

Sirui Hong, Mingchen Zhuge, Jiaqi 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, 2023. URL https://arxiv.org/abs/2308.00352.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for LLM agent reinforcement learning, 2025. URL https://arxiv.org/abs/2509.21240.

Zhida Jiang, Zhaolong Xing, Jiawei Lu, Yipei Niu, Qingyuan Sang, Liangxu Zhang, Wenquan Dai, Junhua Shu, Jiaxing Wang, Qiangyu Pei, Qiong Chen, Xinyu Liu, Fangming Liu, Ai Han, Zhen Chen, and Ke Zhang. Rollout-training co-design for efficient llm-based multi-agent reinforcement learning, 2026. URL https://arxiv.org/abs/2602.09578.

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. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 29557–29590. PMLR, 2025. URL https://proceedings.mlr.press/v267/kazemnejad25a.html. arXiv:2410.01679.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning, 2022. URL https://arxiv.org/abs/2207.01780.

Yizhi Li, Qingshui Gu, Zhoufutu Wen, Ziniu Li, Tianshun Xing, Shuyue Guo, Tianyu Zheng, Xin Zhou, Xingwei Qu, Wangchunshu Zhou, Zheng Zhang, Wei Shen, Qian Liu, Chenghua Lin, Jian Yang, Ge Zhang, and Wenhao Huang. TreePO: Bridging the gap of policy optimization and efficacy and inference efficiency with heuristic tree-based modeling, 2025. URL https://arxiv.org/abs/2508.17445.

Junwei Liao, Muning Wen, Jun Wang, and Weinan Zhang. Marft: Multi-agent reinforcement fine-tuning, 2025. URL https://arxiv.org/abs/2504.16129.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050. arXiv:2305.20050.

Bo Liu, Leon Guertler, Simon Yu, Zichen Liu, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, Wee Sun Lee, and Natasha Jaques. Spiral: Self-play on zero-sum games incentivizes reasoning via multi-agent multi-turn reinforcement learning, 2025a. URL https://arxiv.org/abs/2506.24119.

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

Miles Team. Miles: Enterprise-grade reinforcement learning for large-scale model post-training. https://github.com/radixark/miles, 2026.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Neural Information Processing Systems*, 2022. doi: 10.52202/068431-2011. URL https://arxiv.org/abs/2203.02155. arXiv:2203.02155.

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large language model-based multi-agent collaboration. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2406.07155.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, 2018. URL https://arxiv.org/abs/1803.11485. arXiv:1803.11485.

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.

Ao Sun, Xiaoyu Wang, Zhe Tan, Yu Li, Jiachen Zhu, Yuheng Jia, and Shu Su. CuMA: Aligning LLMs with sparse cultural values via demographic-aware mixture of adapters. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27423–27441, San Diego, California, United States, July 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.1265. URL https://aclanthology.org/2026.acl-long.1265/.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning, 2017. URL https://arxiv.org/abs/1706.05296.

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. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/. arXiv:2312.08935.

Tao Wang, Suhang Zheng, and Xiaoxiao Xu. Rtmc: Step-level credit assignment via rollout trees, 2026. URL https://arxiv.org/abs/2604.11037.

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

Kaiyan Zhang, Kai Tian, Runze Liu, Sihang Zeng, Xuekai Zhu, Guoli Jia, Yuchen Fan, Xingtai Lv, Yuxin Zuo, Che Jiang, Yuru Wang, Jianyu Wang, Ermo Hua, Xinwei Long, Junqi Gao, Youbang

Sun, Zhiyuan Ma, Ganqu Cui, Ning Ding, Biqing Qi, and Bowen Zhou. MARTI: A framework for multi-agent LLM systems reinforced training and inference. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=E7jZqo0A50.

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.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583, 2024. doi: 10.52202/079017-2000. URL https://arxiv.org/abs/2312.07104. arXiv:2312.07104.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. In *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2402.16823.

# A Proofs

## A.1 Proposition 1

Fix a node $v$ with history $s_v$ and acting agent $i = \iota(v)$. All expectations below are conditional on $s_v$ and use the sampling distribution specified in Assumption 1.

**Step 1 (each branch estimates $Q_i^\mu$).** By Assumption 1(ii)–(iii), the $M$ continuations of candidate $a_k$ are i.i.d. draws from $P^\mu(\cdot \mid s_v, a_k)$, and $M$ is chosen before any outcome is observed. Hence from equation 4,

$$\mathbb{E}\big[\widehat{Q}_i(s_v, a_k) \mid s_v, a_k\big] = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}\big[R_i^{(k,m)} \mid s_v, a_k\big] = Q_i^\mu(s_v, a_k).$$

The expectation uses the acting agent's reward $R_i$, so the same argument covers cooperative, zero-sum, and general-sum settings.

**Step 2 (the leave-one-out average estimates $b_i^\mu$).** For the proof, define $\widehat{b}_i^{(-k)} = (K-1)^{-1}\sum_{j\neq k}\widehat{Q}_i(s_v, a_j)$. By Assumption 1(i), each $a_j$ with $j \neq k$ is independent of $a_k$ conditional on $s_v$. By (ii), its reward has conditional mean $Q_i^\mu(s_v, a_j)$ even after conditioning on all candidate actions. Taking expectations over the other candidates therefore gives

$$\mathbb{E}\big[\widehat{b}_i^{(-k)} \mid s_v, a_k\big] = \frac{1}{K-1}\sum_{j\neq k} \mathbb{E}_{a_j\sim\mu}\big[Q_i^\mu(s_v, a_j)\big] = b_i^\mu(s_v).$$

Excluding candidate $k$ gives a baseline whose conditional expectation is $b_i^\mu(s_v)$. Including it instead yields the constant rescaling derived below.

**Step 3 (recovering the group-relative estimator).** The algebra in Appendix A.2 gives $\frac{K}{K-1}\widehat{A}_{i,k} = \widehat{Q}_i(s_v, a_k) - \widehat{b}_i^{(-k)}$, where $\widehat{A}_{i,k}$ is defined in equation 5. Subtracting the conditional expectations in Steps 1 and 2 therefore gives $\mathbb{E}[\frac{K}{K-1}\widehat{A}_{i,k} \mid s_v, a_k] = Q_i^\mu(s_v, a_k) - b_i^\mu(s_v) = A_i^\mu(s_v, a_k)$. □

*Remark* 1 (Role of the execution graph). The graph defines the shared history and the continuation evaluated using $R_{\iota(v)}$. Assumption 1 supplies the conditional-mean identities used in Steps 1 and 2; Proposition 2 establishes these identities for dependency-preserving replay, including correlated rewards from shared saved state.

### A.2 GROUP BASELINE VERSUS LEAVE-ONE-OUT

Write $\widehat{Q}_k = \widehat{Q}_i(s_v, a_k)$ and $S = \sum_{j=1}^{K} \widehat{Q}_j$. The group mean is $\widehat{b}^{\text{group}} = S/K$ and the leave-one-out mean is $\widehat{b}^{\text{loo}(k)} = (S - \widehat{Q}_k)/(K-1)$. Then

$$\widehat{Q}_k - \frac{S}{K} = \frac{K\widehat{Q}_k - S}{K} = \frac{K-1}{K} \cdot \frac{K\widehat{Q}_k - S}{K-1} = \frac{K-1}{K}\left(\widehat{Q}_k - \frac{S - \widehat{Q}_k}{K-1}\right) = \frac{K-1}{K}\left(\widehat{Q}_k - \widehat{b}^{\text{loo}(k)}\right),$$

using $K\widehat{Q}_k - S = (K-1)\widehat{Q}_k - (S - \widehat{Q}_k)$. Here, $\widehat{b}^{\text{loo}(k)}$ is the proof baseline $\widehat{b}_i^{(-k)}$ above. For fixed $K$, the group-relative and leave-one-out signals differ by a common positive factor, preserving signs and candidate ordering. Their advantage-weighted surrogate gradients have the same proportional relationship. Normalizing each signal by its own nonzero group standard deviation cancels this factor when no additive stabilizer is used. Proposition 1 characterizes the raw estimator before this random normalization.

The implementation uses the group mean so that all candidates share one baseline and one standard deviation of the candidate value estimates, fitting the group-relative training procedure directly. Both baseline choices use the same $K$ sampled candidates and retain all candidates for training; the denominator $K-1$ arises from excluding the evaluated candidate only when computing its baseline. The proportional relationship above allows either form to express the same relative preferences.

## B EXPERIMENTAL DETAILS

### B.1 MAS EXECUTION TOPOLOGIES

Figure 5 illustrates the role dependencies in the three MAS settings. Math and code use cooperative verification and revision, with at most one solver or coder revision during training to limit imbalance in role invocation frequencies. Games use alternating decisions by two players with opposing rewards until termination or a game-specific turn limit. Appendix B.3 specifies the evaluation budgets; Appendix B.5 examines whether additional revisions at inference improve math performance.

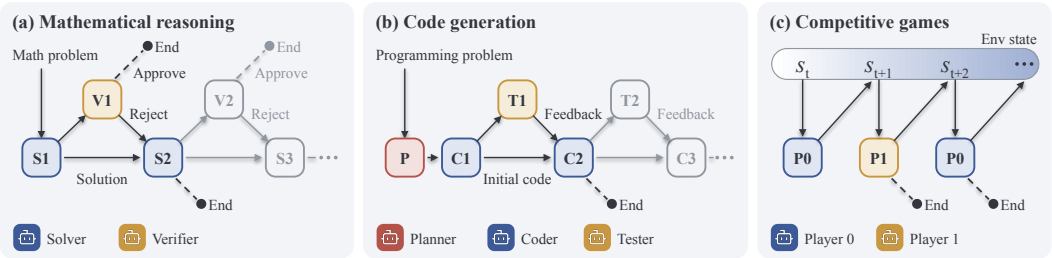


Figure 5: Execution topologies for the three task domains. (a) A verifier can approve the solver's answer or request a revision by the solver. (b) Tester feedback guides code revision. (c) Two players alternate actions as the environment state evolves; $s_t$ denotes the environment state in this panel. Tiles denote model calls, and repeated calls appear as separate nodes; gray nodes indicate extensions beyond the main evaluation settings.

### B.2 TRAINING CONFIGURATION AND BASELINE ADAPTATIONS

All domains use Qwen3-4B with thinking disabled during training and evaluation. Table 2 lists the training hyperparameters. The optimization and batch settings are identical for every method in a domain; methods differ only in how comparison groups are formed and credited. We implement training on Miles (Miles Team, 2026), using SGLang (Zheng et al., 2024) for rollout generation and Megatron for policy optimization, colocated on the same GPUs. The comparisons use final checkpoints after 220 updates for math and games and 162 for code. Equal update and batch budgets do not imply equal generation cost, since a complete comparison must also account for seed discovery, candidate continuations, and discarded groups (Section 4.4).

Table 2: Training hyperparameters. Within each domain, optimization and training sampling settings are shared across methods; the lower block lists comparison-group collection settings for ROLLCREDITS. A dash denotes no top-$k$ truncation.

| | Math | Code | Games |
|---|---|---|---|
| *Optimization (all methods)* | | | |
| Optimizer | Adam, $\beta_1$=0.9, $\beta_2$=0.98, $\epsilon$=$10^{-8}$ | | |
| Learning rate | $10^{-6}$, constant after 20 warm-up updates | | |
| Weight decay / gradient clipping | 0.1 / 1.0 | | |
| Clipping thresholds $\epsilon_l$ / $\epsilon_h$ | 0.2 / 0.28 | | |
| KL and entropy coefficients | 0 / 0 | | |
| Precision | bf16 | | |
| Updates | 220 | 162 | 220 |
| Comparison groups per update | 64 | | |
| Candidates per group $K$ | 8 | | |
| Training sequences per update | 512, one optimizer step | | |
| *Sampling during training* | | | |
| Temperature / top-$p$ / top-$k$ | 1.0 / 1.0 / – | 1.0 / 1.0 / – | 0.7 / 0.8 / 20 |
| Maximum response tokens | 8,192 | 8,192 | 4,096 |
| Context limit | 32,768 | 16,384 | 16,384 |
| Maximum model calls per execution | 3 | 4 | game length |
| *Comparison-group collection* (ROLLCREDITS) | | | |
| Continuations per candidate $M$ | 1 | | |
| Seed executions per task | 2 | | |
| Zero-variance group filtering | yes | | |

**Role selection and reported configurations.** For math, ROLLCREDITS trains the solver's initial and revised answers, while verifier responses participate in the interaction. The selected code configuration trains only coder actions and uses balanced state scheduling; planner and tester responses remain part of the workflow. The selected game configuration uses balanced accepted-group collection and at most two decision states per seed execution. These configurations were selected after inspecting completed evaluations, using MAS pass@8 for math and code and the equal-weight mean return over the three training games for games. Each domain reports all metrics from one configuration's final checkpoint, without selecting a different configuration for each column. The comparisons therefore include configuration selection and do not isolate the effect of any single training change.

**AT-GRPO adaptation.** AT-GRPO samples candidates at a shared environment state for each agent and turn, scores them using task-specific feedback, and advances the interaction with the highest-scoring candidate. In our math implementation, local feedback checks the correctness of the solver's initial and revised answers and the verifier's assessment. ROLLCREDITS estimates each candidate's expected outcome reward through its own complete continuation under the collection policy.

### B.3 EVALUATION PROTOCOLS

For the main comparison in Table 1, MAS inference retains the training revision budget in math and code and the same environment turn limits in games. The extended math revision-budget experiment is reported separately in Appendix B.5.

**Mathematical reasoning.** We use all 500 MATH500 problems, 30 AIME24 problems, 30 AIME25 problems, and 40 AMC23 problems. Each problem receives eight sampled responses or complete MAS executions, with at most three model calls in MAS inference (solver, verifier, and one solver revision). Sampling uses temperature 0.7, top-$p$ 0.8, top-$k$ 20, and evaluation seed 42, with thinking disabled. The evaluation context limit is 32,768 tokens, with output limits of 8,192 tokens for solver responses and 2,048 for verifier responses. Training and evaluation use the same answer grader. Pass@1 is the mean correctness of the eight samples; pass@8 is the fraction of problems solved by at least one sample. We report an equal-weight macro average over the four benchmarks.

**Code generation.** We evaluate on 164 HumanEval+ problems, 378 MBPP+ problems, and 1,055 LiveCodeBench problems from `release_v6`, without date filtering. Execution-based grading uses EvalPlus and SandboxFusion. Each problem receives eight samples under both single-agent and MAS inference, using the same sampling parameters as math. MAS inference allows one coder revision after planner, coder, and tester calls, for at most four model calls. The context limit is 16,384 tokens; output limits are 8,192 for single-agent and coder responses, 4,096 for planner responses, and 2,048 for tester responses. Aggregate pass@1 and pass@8 are micro averages over all 1,597 problems. The AT-GRPO comparison uses the variant trained without an additional intermediate reward.

**Games.** In *Colonel Blotto*, players secretly allocate 20 units across three battlefields; allocations are revealed together, and winning more battlefields wins the round. The first player to win three rounds wins the match, with at most five rounds. In *Connect Four*, players take turns dropping a piece into a column of a $6 \times 7$ board, aiming to form a horizontal, vertical, or diagonal line of four. In *Liar's Dice*, each player initially holds five private dice and raises public bids about the combined dice or challenges the current bid; the loser of a challenge loses a die, and losing all dice ends the game. In held-out *Breakthrough*, players advance pieces on a $6 \times 6$ board, capturing opponents diagonally and seeking to reach the opposite back rank or eliminate the opposing pieces.

We use TextArena 0.7.3 with 256 paired seeds per environment, starting from seed 42. Each seed is evaluated with the candidate in both seats against frozen Qwen3-4B, giving 512 games per environment. Sampling follows the math parameters, with a 16,384-token context limit, a 4,096-token response limit, and prompts that list the legal actions. Win rate is the fraction of games won, with draws included in the denominator and equal weight on the two seats. Returns are averaged over both seats on the original $[-1, 1]$ scale. The in-distribution (ID) macro average weights Colonel Blotto, Connect Four, and Liar's Dice equally; Breakthrough is held out and reported separately.

The environment caps valid turns at 10 for Colonel Blotto, 42 for Connect Four, 64 for Liar's Dice, and 108 for Breakthrough. Games end earlier when a terminal outcome is reached; reaching the cap without one yields a draw with zero return.

Breakthrough uses a fixed initial position on a $6 \times 6$ board, with diagonal moves allowed only for captures. The paired seeds therefore sample stochastic play rather than distinct starting positions. Reported outcomes include losses due to invalid actions. The environment also has a terminal-handling edge case in which a player with no legal moves is asked to act and eventually loses through the invalid-action rule. These outcomes remain in the reported protocol; the scores should be interpreted as performance under that environment implementation.

**Uncertainty estimates.** Math and code intervals use a percentile bootstrap with 20,000 resamples. Resampling is stratified by benchmark and uses problems as the sampling unit, keeping all eight generated samples for each problem together. Each resample recomputes pass@1 and pass@8 and then the math macro average or code micro average. Methods and inference modes use the same problem-resampling indices within each benchmark. Game intervals use paired bootstrap resampling over seed pairs, preserving both seats with 20,000 resamples. The supplementary tables report marginal 95% intervals, not intervals for method differences or significance tests. Each configuration has one training seed; the intervals condition on the checkpoint and generated samples and do not include variation from retraining, regeneration, or post-hoc configuration selection.

### B.4 Complete task results

Tables 3 and 4 expand the math and code aggregates in Table 1 into benchmark-level results for both inference modes and both pass metrics. They also retain the domain aggregates and their confidence intervals. Table 5 gives each game's win rate and return. GRPO denotes single-agent training, and traj-GRPO denotes trajectory-level MAS training; Single and MAS specify the inference protocol. Math/code scores and game win rates are reported to two decimal places here, versus one in Table 1; returns retain three decimals. All intervals use the protocols in Appendix B.3 and condition on the reported checkpoints and generated samples.

**Benchmark-level mathematical reasoning.** RollCredits has the highest single-agent pass@1 on MATH500, AIME24, and AIME25 (Table 3). On MATH500, it leads in pass@1 under both

Table 3: Complete mathematical reasoning results (%). Each benchmark uses eight samples per problem under Single and MAS inference; Macro weights the four benchmarks equally. Intervals use a problem-level stratified bootstrap. Brackets give the lower and upper limits of the 95% confidence interval. Base is the initial Qwen3-4B checkpoint before our RL training. Bold marks the highest point estimate among the trained methods in each row, without implying statistical significance.

| Metric | Base | GRPO | traj-GRPO | Dr. MAS | AT-GRPO | RollCredits |
|---|---|---|---|---|---|---|
| *MATH500 (500)* | | | | | | |
| Single pass@1 | 81.50 [78.62, 84.25] | 90.60 [88.33, 92.70] | 88.53 [86.00, 90.88] | 91.00 [88.73, 93.10] | 91.28 [89.10, 93.28] | **91.50** [89.30, 93.53] |
| Single pass@8 | 92.40 [90.00, 94.60] | **95.80** [94.00, 97.40] | 93.20 [91.00, 95.40] | **95.80** [94.00, 97.40] | **95.80** [94.00, 97.40] | **95.80** [94.00, 97.40] |
| MAS pass@1 | 82.05 [79.20, 84.78] | 91.00 [88.75, 93.08] | 92.40 [90.40, 94.28] | 92.23 [90.18, 94.10] | 91.75 [89.63, 93.70] | **92.45** [90.40, 94.30] |
| MAS pass@8 | 93.20 [90.80, 95.40] | 96.00 [94.20, 97.60] | 96.60 [94.80, 98.00] | **97.00** [95.40, 98.40] | 96.00 [94.20, 97.60] | 96.40 [94.60, 98.00] |
| *AIME24 (30)* | | | | | | |
| Single pass@1 | 20.83 [10.00, 32.92] | 41.67 [26.67, 57.08] | 37.50 [23.33, 52.08] | 47.50 [33.75, 61.67] | 41.67 [27.08, 56.67] | **47.92** [34.58, 61.67] |
| Single pass@8 | 46.67 [30.00, 63.33] | 63.33 [46.67, 80.00] | 56.67 [40.00, 73.33] | 73.33 [56.67, 90.00] | 66.67 [50.00, 83.33] | **76.67** [60.00, 90.00] |
| MAS pass@1 | 23.75 [12.08, 36.67] | 40.83 [25.83, 56.67] | 54.58 [40.41, 68.75] | **60.42** [47.08, 73.33] | 47.08 [31.67, 62.50] | 55.42 [41.25, 69.58] |
| MAS pass@8 | 50.00 [33.33, 66.67] | 60.00 [43.33, 76.67] | **80.00** [66.67, 93.33] | **80.00** [66.67, 93.33] | 70.00 [53.33, 86.67] | **80.00** [66.67, 93.33] |
| *AIME25 (30)* | | | | | | |
| Single pass@1 | 20.00 [9.17, 32.08] | 39.17 [25.00, 54.17] | 27.08 [13.33, 42.08] | 31.25 [17.92, 45.42] | 37.50 [23.75, 51.67] | **39.58** [25.00, 54.58] |
| Single pass@8 | 36.67 [20.00, 53.33] | 60.00 [43.33, 76.67] | 43.33 [26.67, 60.00] | 50.00 [33.33, 66.67] | **63.33** [46.67, 80.00] | 56.67 [40.00, 73.33] |
| MAS pass@1 | 18.33 [7.92, 30.42] | 40.00 [25.83, 55.00] | **48.33** [33.33, 63.33] | 47.92 [33.33, 62.92] | 40.42 [27.07, 54.58] | 45.42 [30.83, 60.00] |
| MAS pass@8 | 36.67 [20.00, 53.33] | 60.00 [43.33, 76.67] | 66.67 [50.00, 83.33] | 63.33 [46.67, 80.00] | **70.00** [53.33, 86.67] | **70.00** [53.33, 86.67] |
| *AMC23 (40)* | | | | | | |
| Single pass@1 | 66.56 [55.00, 77.81] | **89.69** [81.88, 95.94] | 75.00 [63.13, 85.94] | 82.81 [72.50, 91.88] | 88.13 [80.00, 94.69] | 86.88 [78.44, 93.75] |
| Single pass@8 | 90.00 [80.00, 97.50] | **97.50** [92.50, 100.00] | 85.00 [72.50, 95.00] | 92.50 [82.50, 100.00] | **97.50** [92.50, 100.00] | 95.00 [87.50, 100.00] |
| MAS pass@1 | 69.38 [58.13, 80.00] | 89.69 [81.56, 96.25] | 90.00 [82.19, 96.25] | **92.19** [85.31, 97.19] | 89.38 [80.63, 96.56] | 90.94 [83.44, 96.88] |
| MAS pass@8 | 92.50 [82.50, 100.00] | 95.00 [87.50, 100.00] | **97.50** [92.50, 100.00] | **97.50** [92.50, 100.00] | 95.00 [87.50, 100.00] | **97.50** [92.50, 100.00] |
| *Macro (4 benchmarks)* | | | | | | |
| Single pass@1 | 47.22 [42.33, 52.41] | 65.28 [59.71, 70.99] | 57.03 [51.25, 63.08] | 63.14 [57.66, 68.74] | 64.64 [59.27, 70.26] | **66.47** [61.19, 71.90] |
| Single pass@8 | 66.43 [59.71, 73.24] | 79.16 [72.70, 85.42] | 69.55 [62.63, 76.49] | 77.91 [71.49, 84.16] | 80.83 [74.56, 86.96] | **81.03** [74.88, 87.02] |
| MAS pass@1 | 48.38 [43.42, 53.57] | 65.38 [59.81, 71.18] | 71.33 [65.80, 76.91] | **73.19** [67.95, 78.41] | 67.16 [61.59, 72.88] | 71.06 [65.70, 76.44] |
| MAS pass@8 | 68.09 [61.52, 74.75] | 77.75 [71.13, 84.26] | 85.19 [79.36, 90.77] | 84.46 [78.57, 90.09] | 82.75 [76.60, 88.68] | **85.98** [80.19, 91.35] |

Table 4: Complete code generation results (%). Each benchmark uses eight samples per problem under Single and MAS inference; Micro weights all 1,597 problems equally. Intervals use a problem-level stratified bootstrap. Brackets give the lower and upper limits of the 95% confidence interval. Base is the initial Qwen3-4B checkpoint before our RL training. Bold marks the highest point estimate among the trained methods in each row, without implying statistical significance.

| Metric | Base | GRPO | traj-GRPO | Dr. MAS | AT-GRPO | RollCredits |
|---|---|---|---|---|---|---|
| *HumanEval+ (164)* | | | | | | |
| Single pass@1 | 75.99 [70.12, 81.63] | **81.33** [76.14, 86.28] | 76.14 [70.35, 81.71] | 78.73 [73.25, 83.99] | 76.75 [70.88, 82.39] | 80.79 [75.53, 85.75] |
| Single pass@8 | 87.20 [81.71, 92.07] | 91.46 [87.20, 95.12] | 87.20 [81.71, 92.07] | 90.24 [85.37, 94.51] | 87.80 [82.93, 92.68] | **92.07** [87.80, 95.73] |
| MAS pass@1 | 79.19 [73.70, 84.38] | **82.77** [77.74, 87.42] | 80.41 [75.00, 85.52] | 79.88 [74.70, 84.83] | 82.01 [76.91, 86.74] | 82.32 [77.29, 86.89] |
| MAS pass@8 | 88.41 [83.54, 93.29] | 92.07 [87.80, 95.73] | 89.02 [84.15, 93.29] | 92.68 [88.41, 96.34] | 91.46 [87.20, 95.73] | **94.51** [90.85, 97.56] |
| *MBPP+ (378)* | | | | | | |
| Single pass@1 | 61.90 [57.21, 66.47] | 62.76 [58.10, 67.29] | 62.27 [57.64, 66.87] | 62.17 [57.57, 66.73] | 63.36 [58.80, 67.92] | **65.24** [60.75, 69.68] |
| Single pass@8 | 69.58 [64.81, 74.07] | 70.90 [66.40, 75.40] | 70.37 [65.61, 74.87] | 71.69 [67.20, 76.19] | 71.96 [67.46, 76.46] | **73.28** [68.78, 77.78] |
| MAS pass@1 | 65.18 [60.75, 69.61] | 66.07 [61.64, 70.47] | 65.11 [60.65, 69.54] | 63.53 [59.09, 67.96] | 65.54 [61.08, 69.91] | **66.70** [62.30, 71.10] |
| MAS pass@8 | 73.28 [68.78, 77.78] | **76.19** [71.96, 80.42] | 73.54 [69.05, 78.04] | 72.75 [68.25, 77.25] | 75.13 [70.63, 79.37] | 75.13 [70.63, 79.37] |
| *LiveCodeBench (1,055)* | | | | | | |
| Single pass@1 | 36.90 [34.22, 39.60] | **49.34** [46.68, 51.99] | 40.01 [37.38, 42.65] | 40.66 [38.03, 43.35] | 40.75 [38.10, 43.44] | 48.91 [46.28, 51.55] |
| Single pass@8 | 46.64 [43.60, 49.67] | **64.36** [61.52, 67.20] | 54.22 [51.18, 57.25] | 55.36 [52.32, 58.29] | 54.88 [51.85, 57.91] | 64.17 [61.23, 67.01] |
| MAS pass@1 | 40.20 [37.58, 42.87] | 50.68 [48.03, 53.35] | 45.41 [42.77, 48.10] | 46.01 [43.34, 48.66] | 47.33 [44.70, 50.02] | **54.29** [51.67, 56.91] |
| MAS pass@8 | 54.31 [51.37, 57.35] | 65.59 [62.75, 68.44] | 59.62 [56.68, 62.56] | 60.19 [57.25, 63.13] | 62.37 [59.43, 65.31] | **68.91** [66.07, 71.66] |
| *Micro (1,597)* | | | | | | |
| Single pass@1 | 46.83 [44.64, 48.98] | 55.80 [53.67, 57.93] | 48.99 [46.82, 51.14] | 49.66 [47.51, 51.83] | 49.80 [47.65, 51.97] | **56.05** [53.93, 58.16] |
| Single pass@8 | 56.23 [53.91, 58.55] | 68.69 [66.44, 70.88] | 61.43 [59.05, 63.74] | 62.81 [60.49, 65.12] | 62.30 [59.99, 64.62] | **69.19** [66.94, 71.38] |
| MAS pass@1 | 50.12 [47.97, 52.22] | 57.62 [55.49, 59.71] | 53.67 [51.53, 55.79] | 53.63 [51.53, 55.74] | 55.21 [53.09, 57.33] | **60.10** [58.02, 62.16] |
| MAS pass@8 | 62.30 [59.99, 64.62] | 70.82 [68.63, 73.01] | 65.94 [63.62, 68.19] | 66.50 [64.25, 68.75] | 68.38 [66.12, 70.57] | **73.01** [70.88, 75.14] |

Table 5: Complete game results against frozen Qwen3-4B: seat-balanced win rates (%) and returns on the $[-1, 1]$ scale. Draws count in the win-rate denominator. Colonel Blotto, Connect Four, and Liar's Dice are training environments; Breakthrough is held out. Intervals use a paired bootstrap over seed pairs. Brackets give the lower and upper limits of the 95% confidence interval. Bold marks the highest point estimate in each row, without implying statistical significance.

| Metric | GRPO | traj-GRPO | Dr. MAS | AT-GRPO | ROLLCREDITS |
|---|---|---|---|---|---|
| *Colonel Blotto (ID)* | | | | | |
| Win rate | 16.60 [13.67, 19.73] | 25.98 [22.27, 29.88] | 93.75 [91.60, 95.70] | 67.38 [64.06, 70.70] | **99.02** [98.05, 99.80] |
| Return | −0.664 [−0.723, −0.604] | −0.207 [−0.279, −0.133] | +0.914 [+0.885, +0.943] | +0.518 [+0.463, +0.572] | **+0.988** [+0.977, +0.998] |
| *Connect Four (ID)* | | | | | |
| Win rate | 56.84 [53.52, 59.96] | 53.71 [50.00, 57.42] | 62.50 [58.79, 66.41] | **65.82** [62.30, 69.34] | 60.35 [56.45, 64.26] |
| Return | +0.137 [+0.074, +0.199] | +0.074 [+0.000, +0.148] | +0.250 [+0.176, +0.324] | **+0.316** [+0.246, +0.387] | +0.207 [+0.125, +0.285] |
| *Liar's Dice (ID)* | | | | | |
| Win rate | 54.10 [50.00, 58.20] | 23.05 [19.53, 26.76] | 61.13 [56.84, 65.43] | 60.94 [56.64, 65.23] | **75.78** [72.07, 79.49] |
| Return | +0.082 [+0.000, +0.168] | −0.539 [−0.609, −0.465] | +0.223 [+0.137, +0.309] | +0.219 [+0.137, +0.305] | **+0.516** [+0.445, +0.586] |
| *Breakthrough (OOD)* | | | | | |
| Win rate | 24.61 [20.90, 28.52] | 58.59 [54.30, 62.89] | 70.31 [66.41, 74.02] | 55.27 [50.98, 59.77] | **84.77** [81.84, 87.70] |
| Return | −0.508 [−0.582, −0.434] | +0.172 [+0.090, +0.258] | +0.406 [+0.328, +0.480] | +0.105 [+0.016, +0.191] | **+0.695** [+0.633, +0.754] |

protocols, at 91.50% for Single and 92.45% for MAS. Its highest aggregate MAS pass@8 combines ties for the best score on AIME24, AIME25, and AMC23; Dr. MAS leads on MATH500. For the same ROLLCREDITS checkpoint, MAS inference raises both metrics on every benchmark. The pass@8 increase is largest on AIME25, from 56.67% to 70.00%, compared with 95.80% to 96.40% on MATH500. MATH500 and AMC23 already have single-agent pass@8 of at least 95%, leaving less room for an increase.

**Benchmark-level code generation.** Under MAS inference, ROLLCREDITS leads in pass@1 on MBPP+ and LiveCodeBench and in pass@8 on HumanEval+ and LiveCodeBench (Table 4). Its largest pass@1 gain over GRPO is on LiveCodeBench, from 50.68% to 54.29%, while MBPP+ rises from 66.07% to 66.70% and HumanEval+ remains close at 82.32% versus 82.77%. LiveCodeBench contains 1,055 of the 1,597 problems and therefore contributes most to this micro-average gain. For pass@8, the gains on HumanEval+ and LiveCodeBench outweigh the lower MBPP+ score of 75.13% versus GRPO's 76.19%. The aggregate improvements thus reflect different benchmark contributions for pass@1 and pass@8.

### B.5 REVISION BUDGET

We test whether solver–verifier cooperation learned within the training interaction budget remains effective over additional rounds at inference. All trained policies use at most one solver revision per execution during training. We evaluate each final math checkpoint with the limit raised to $k = 3$, so that a rejected revision returns to the verifier; the execution ends when the verifier approves or the budget is exhausted. Budgets $k = 1$ and $k = 2$ are obtained by truncating each execution after its first or second revision and grading the answer at that point, so all budgets share the same samples. Other settings follow the math protocol in Appendix B.3. Because this evaluation is an independent regeneration, we report each method's change relative to its own $k = 1$ score.

Table 6 gives paired differences with bootstrap intervals that use the same problem resampling across budgets and methods. Raising the limit to $k = 3$ does not lower the macro pass@1 of any method. For ROLLCREDITS, each added revision gives a significant gain, totaling 0.82 percentage points

Table 6: Change in math MAS pass@1 (points, macro average of four benchmarks) when the revision limit is raised at inference. Trained policies use $k = 1$ during training; brackets give paired 95% bootstrap intervals.

| Method | $k = 1 \to 2$ | $k = 2 \to 3$ | $k = 1 \to 3$ |
|---|---|---|---|
| Base | +0.30 [+0.04, +0.64] | +0.04 [+0.00, +0.08] | +0.34 [+0.08, +0.68] |
| GRPO | +0.32 [+0.03, +0.67] | −0.09 [−0.49, +0.23] | +0.23 [−0.15, +0.63] |
| traj-GRPO | −0.02 [−1.48, +1.45] | +1.19 [−0.15, +2.64] | +1.18 [−0.44, +2.81] |
| Dr. MAS | +0.40 [−0.66, +1.43] | +0.53 [−0.12, +1.26] | +0.94 [−0.34, +2.26] |
| AT-GRPO | +0.42 [+0.10, +0.85] | −0.08 [−0.23, +0.00] | +0.34 [−0.05, +0.79] |
| ROLLCREDITS | +0.43 [+0.10, +0.85] | +0.39 [+0.08, +0.81] | +0.82 [+0.29, +1.46] |

from $k = 1$ to $k = 3$, and its pass@1 does not decrease with $k$ on any of the four benchmarks. These results support the ability of the trained solver–verifier workflow to sustain useful cooperation beyond the training interaction budget, improving performance through additional verification and revision.

### B.6 CREDIT-QUALITY DIAGNOSTIC

The diagnostic in Section 4.3 freezes the reported checkpoints. Histories and uncertainty rules are fixed before measurement; histories are taken from saved executions in a fixed order and are not selected by any estimator's score.

**Credit-source comparison.** Figure 3(a,b) uses 48 math solver histories followed by further interaction (8 each from MATH500 and AMC23 and 16 each from AIME24 and AIME25) and 24 late-turn game states (four per training game and seat). Each history has two fixed candidate actions, each evaluated in eight repeated groups with seven comparison actions resampled per group ($K$=8). Independent references use 32 continuations per fixed action to estimate $Q_i^\mu$ and 32 policy samples per history to estimate $b_i^\mu$. Games share the states and first two candidate actions from Figure 3(c), but use new comparison groups and policy-baseline samples independent of that panel's measurement pool.

All credit sources use the acting agent's reward and are evaluated against $Q_i^\mu(s_v, a) - b_i^\mu(s_v)$. Local credit subtracts the seven comparison actions' mean local score from the fixed action's local score. Trajectory-level credit subtracts the mean reward of seven independent executions from the task's initial state from the fixed action's first terminal return. Continuation credit subtracts the seven comparison actions' mean estimated value from the fixed action's estimated value, using the leave-one-out baseline in Appendix A.2.

Continuation budgets are compared within the same states, candidate groups, and references. Math uses $M$=1, 4, retaining existing draws and adding those needed for $M$=4. Games use $M$=4, 16, retaining the first four draws and adding 12 independent continuations per candidate. The game follow-up budget was chosen after inspecting $M$=4 and fixed before further sampling. Matched game $M$=1 estimates retain only the first recorded continuation per candidate.

For each fixed action, variance is estimated across the eight repeats. Squared bias subtracts the finite-repeat mean variance and independent reference variance from the squared mean error; MSE subtracts the reference variance from the mean squared error. Results are averaged over actions and histories. Noise-corrected estimates are left untruncated and can fall below zero without implying negative population squared bias.

In games, increasing $M$ from 4 to 16 reduces MSE by 0.119 (95% paired bootstrap CI $[0.079, 0.167]$). The MSE difference from local feedback is 0.011 ($[-0.008, 0.028]$). At the training budget $M$=1, matched estimates of squared bias, variance, and MSE are 0.012, 0.578, and 0.589, respectively.

**Continuation budget and estimation error.** Figure 3(c) isolates continuation-sampling error by centering advantages over four fixed candidates, rather than estimating the policy baseline as in panels (a,b). Each state has four policy-sampled candidate actions, a reference of 32 continuations

Table 7: Saved-state reuse at 32 fixed decision states ($K = 8$, $M = 1$). Rebuilt uses ancestor replay to reconstruct each candidate's history before its group; Saved starts candidates from the saved state. Medians over three repetitions, with ranges in brackets. Recomputed prompt tokens (thousands) are those not served from the prefix cache.

| | Math | | Games | |
|---|---|---|---|---|
| | Rebuilt | Saved | Rebuilt | Saved |
| Calls | 655 | 400 | 4,563 | 2,693 |
| Recomputed tokens | 1,056 [1,033, 1,084] | 806 [794, 836] | 1,181 [1,176, 1,183] | 888 [886, 978] |
| Reduction | | 23.7% | | 24.8% |

per candidate, and a separate pool of $N$ continuations per candidate. The expected squared error of an $M$-continuation estimate of the advantage centered over the four fixed candidate actions at that state is computed in closed form as $\beta^2 + \sigma^2/M$, where $\beta^2$ is the noise-corrected squared bias of the pool mean and $\sigma^2$ is the per-continuation variance; no Monte Carlo subsampling is involved. The math curve uses 24 solver states with $N$=32, six per benchmark, taken from the same pool in its natural snapshot order without outcome-based selection. The game curve uses 24 states with $N$=32, four per game and seat, taken at the first time the acting seat reaches a predetermined late turn: the second move in Colonel Blotto and the sixth move in Connect Four and Liar's Dice. Of 56 candidate executions, 40 reached this turn. Local feedback is nonzero at only 2 of the 24 states; elsewhere its error reflects differences in action value that a zero score ignores, and 15 states contribute positive local error. The crossover $M$ is $\sigma^2/(\mathrm{MSE}_{\mathrm{local}} - \beta^2)$.

**Additional diagnostic state sets.** Three additional state sets were evaluated. A second game state set included late states whose screened value satisfied $|\widehat{Q}| \geq 0.5$, using continuations disjoint from the measurement; on these 12 states local feedback had squared error 0.012 with an interval including zero, and the crossover lies at $M{\approx}43$. This selection favors states whose outcome is already largely decided, where every action has similar value. At 24 early-game histories, local feedback coincides with the zero-credit control, and $K$=4, $M$=2 reduces the squared error of $K$=8, $M$=1 from 1.02 to 0.63, with a 95% interval for the difference of $[-0.58, -0.22]$. On 12 intermediate code histories, all sources have squared error below 0.01, because downstream roles rarely change the final grading outcome.

**Uncertainty.** Intervals are 95% percentile intervals from a bootstrap that resamples complete histories with 20,000 draws, keeping fixed candidate actions, baselines, and repeats of all sources together. They describe variation across the sampled histories, are not corrected for multiple comparisons, and are exploratory given the number of histories.

### B.7 Collection measurements

**Saved-state reuse.** We measure the recomputation avoided by saved states on fixed inputs. For each of math and games, we take the final RollCredits checkpoint and select 32 saved decision states found by the training collector's state discovery on training tasks, balanced across agents (math) or games and seats (games). Each group generates $K = 8$ candidate actions and $M = 1$ continuation per candidate with the training sampling settings, and no parameters are updated. All runs use one SGLang server with the same configuration, and the prefix cache is flushed before each run. The training collector starts every candidate from its saved state. The comparison collector instead uses ancestor replay to rebuild each candidate's history from the task's initial state before its group, sending every ancestor turn to the server with its saved action so that it reaches the same state. Both collectors run with the same concurrency and share the prefix cache. Each is repeated three times in rotated order, and Table 7 reports medians with ranges.

**Prefix-cache reuse.** Figure 4 uses two per-update metrics logged for every training run: the fraction of prompt tokens served from the inference backend's prefix cache and the mean number of cached prompt tokens per training sample, both aggregated over the samples used for the policy update. Their ratio gives the mean prompt length, and the difference gives the uncached prompt tokens per sample. We average over all updates of the runs that produced the checkpoints in Table 1,

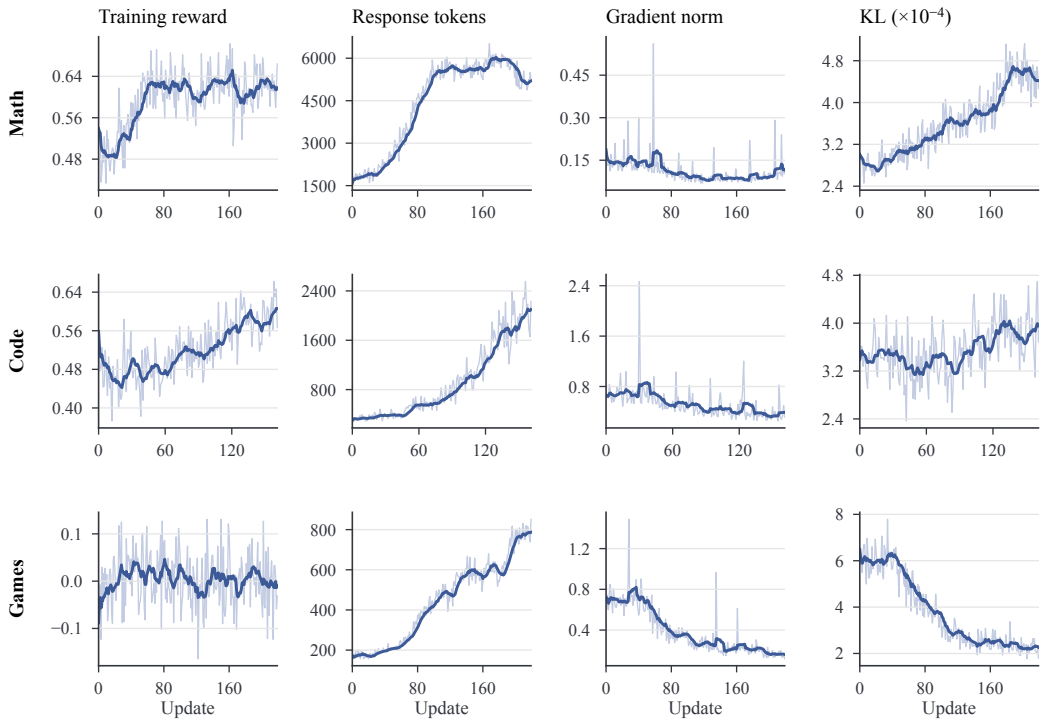


Figure 6: Training dynamics of the ROLLCREDITS runs in Table 1. Columns show the mean reward of the training samples, the mean response length in tokens, the gradient norm, and the mean per-token KL divergence between the log-probabilities of the training and inference engines on the same samples. Light lines are per-update values and dark lines trailing 10-update means. Game rewards are self-play returns and are zero-sum across seats.

chaining resumed runs by update index. Every method covers all updates except code traj-GRPO, whose logged history starts at update 20 after a resumption. Samples generated only as continuations, whose rewards feed the advantage estimate but which are not used as training sequences, may not be included; the counts therefore describe the reuse of shared histories by training samples rather than the total generation cost of collection.

### B.8 TRAINING DYNAMICS

Figure 6 summarizes the runs in Table 1. Training rewards rise in math and code. Game rewards are self-play returns and remain near zero when averaged across opposing seats; performance is evaluated against the frozen initial policy. Response length increases across domains, approaching the 8,192-token limit in math. Gradient norms show no sustained growth. The discrepancy between training- and inference-engine log-probabilities remains below $10^{-3}$, indicating numerical consistency between the engines.

## C SAMPLING AND IMPLEMENTATION DETAILS

### C.1 REPLAY AND CONDITIONAL SAMPLING

Downstream replay preserves the ancestor history and regenerates calls and environment transitions affected by the candidate action. Reuse is valid when the recorded dependencies include all inputs affecting these transitions and their rewards. Let $Z$ denote additional saved information, such as environment state or unaffected parallel outputs, and let $\nu(\mathrm{d}z \mid s_v)$ be its conditional law under the target execution process. The following result relates these execution rules to the advantage estimator.

**Proposition 2** (Dependency-preserving replay)**.** *Fix $s_v$ and suppose the execution law factors into conditional call and environment-transition kernels, with finite termination and integrable rewards. Suppose (i) saved states follow $\nu(\cdot \mid s_v)$, and candidates are sampled i.i.d. from $\mu(\cdot \mid s_v)$ independently of $Z$; (ii) saved outputs are unaffected by the candidate action, with their dependencies fixed by $s_v$ or included in $Z$; and (iii) replay samples all remaining calls and transitions from their target conditional kernels, using fresh randomness, the policy $\mu$, and the updated dependencies. For a fixed $M$, shared-state replay satisfies*

$$\mathbb{E}\Big[\widehat{Q}_i(s_v, a_j) \mid s_v, a_1, \ldots, a_K\Big] = Q_i^{\mu}(s_v, a_j), \qquad j = 1, \ldots, K.$$

*Consequently, the corrected group estimator has the conditional expectation in Proposition 1, including when all candidates share $Z$.*

*Proof.* Conditional on $(s_v, z, a_j)$, replay restores the unaffected dependencies and applies the target kernel at each remaining call or environment transition. Induction over the execution order gives the target conditional continuation law, including candidate-dependent calls and termination. Write its expected reward as $q_i^{\mu}(s_v, a_j, z)$. Since the saved-state law conditional on all candidate actions remains $\nu(\cdot \mid s_v)$, averaging over $Z$ gives

$$\begin{aligned}&\mathbb{E}\Big[\widehat{Q}_i(s_v, a_j) \mid s_v, a_1, \ldots, a_K\Big] \\ &\qquad = \int q_i^{\mu}(s_v, a_j, z)\, \nu(\mathrm{d}z \mid s_v) = Q_i^{\mu}(s_v, a_j).\end{aligned}$$

Independent candidate sampling then gives the baseline mean in Step 2 of Appendix A; the same subtraction and group-size correction complete the argument. □

For parallel workers followed by an aggregator, replay regenerates the aggregator's response to the candidate worker output. Another worker's unaffected output may be shared within the group when it follows the required conditional law. The expectation above averages over that output across saved states; fixing one particular output instead gives credit conditional on that additional information. Shared state can correlate rewards while preserving the conditional means needed by the proof.

The role instructions and observations used to sample candidates are included in $s_v$ through $o_v$ in equation 1; additional saved information is represented by $Z$. State selection must also preserve the conditional law of $Z$ given that history. These conditions connect the saved environment state to the history used for credit assignment.

**Sampling error.** Complete continuations avoid approximation error from predicting the reward of an unfinished execution, while retaining Monte Carlo sampling noise. With finite conditional reward variance, averaging more independent continuations reduces the variance of each candidate's value estimate at additional generation cost. The proposition establishes the target of the raw credit estimate; comparisons of estimation error and training efficiency also depend on the rollout budget and the accuracy of alternative value estimators.

### C.2 Training signals and policy updates

The shared model takes different role instructions and observations as input. For the roles selected for training, agent-specific advantages supply learning signals based on the corresponding agent's reward, so experiences from different roles contribute to a common set of parameters.

*Remark* 2 (Training signal). Training uses $\widehat{A}_{i,k}$ without the fixed correction, which preserves signs and candidate ordering for a fixed $K$. Group-relative standardization and filtering are applied in the optimization procedure of Section 3.3; the proposition gives the conditional expectation of the raw estimate after the fixed group-size correction.

Appendix A.2 derives the fixed proportional relationship between the group-relative signal and the corrected estimator. Standardization divides the signal by the standard deviation of the candidate value estimates within the group, changing the relative weight of groups. Filtering retains groups with nonidentical candidate value estimates and changes the training distribution. These operations

are applied after the raw credit estimate characterized by Proposition 1; matched comparisons track their settings and group acceptance rates.

The advantage evaluates a local action deviation under $\mu$. With parameters shared across roles, an update can change several agents' policies simultaneously. The decision-level estimand therefore provides no guarantee of joint improvement or convergence to an equilibrium.

### C.3 Scheduling and policy versions

Decision states can be made available during seed generation or selected after the seed completes. Candidate branches, states from one seed, and multiple seeds are dispatched to a shared inference pool with a limit on outstanding requests. Separate environment copies prevent one branch from mutating another's state. Decision states from the same seed can remain statistically dependent even when their branches are scheduled concurrently. Total generation cost includes seed discovery, candidate actions, and downstream calls; selecting more states from a seed allocates more of this budget to histories visited by that seed.

Candidate actions and continuations use one collection policy $\mu$ throughout each group, including when branches execute concurrently.

The ratio in equation 6 uses the probability associated with the policy that generated each token. Policy-version records and sampling log-probabilities identify the collection policy and the denominator used in training.

## D Role prompts

Every role is played by the same policy. Each prompt is sent as a single user message through the Qwen3 chat template with thinking disabled. Placeholders in braces are filled with the task statement and the outputs of earlier roles in the same execution; public example executions and sandbox reports are inserted as JSON.

### D.1 Mathematical reasoning

**Solver, initial answer**

```
# Task Introduction
{problem}

# Your Role
You are a "Solver Agent". Please reason step by step, and put your final answer within \
boxed{}.
```

**Verifier**

```
# Task Introduction
{problem}

# Solver Solution
{solver_output}

# Your Role
You are a "Verifier Agent". Your responsibility is to critically review the most recent
solution from the "Solver Agent".
Check the mathematical reasoning and the final answer. Reject only substantive mathematical
 errors or missing conditions that can change the final answer; do not reject merely for
style, brevity, or harmless formatting.
At the end of your output, you MUST provide exactly one verdict within <verify> </verify>:
(1) <verify>approve</verify> if the reasoning and final answer are correct.
(2) <verify>reject</verify> if the solution contains a substantive mathematical error.

If you reject, first provide a concise, actionable diagnosis inside <critique> </critique>,
 then output <verify>reject</verify>.
Responses without exactly one parseable verdict are treated as reject.
```

**Solver, revision**

```
# Task Introduction
{problem}

# Solver Solution
{solver_output}

# Verifier Feedback
Parsed verifier verdict: {verifier_verdict}
{verifier_output}

# Your Role
You are a "Solver Agent". Your job is to revise your latest solution after verifier
feedback.
Treat the verifier critique as a hypothesis, independently check the criticized step,
preserve correct reasoning, and revise only when the critique identifies a real
mathematical error or missing condition.
Please reason step by step, and put your final answer within \boxed{}.
```

### D.2 CODE GENERATION

**Planner**

```
# Task Introduction
{problem}

# Your Role
You are a "Planner Agent". Produce a concise implementation plan for the Coder Agent.
Identify the required input/output contract, algorithm, important edge cases, and
complexity.
Do not write the final program. Do not invent requirements, tests, or tool results that are
 not present in the task.
No execution tool is available in this turn. Do not claim that you ran code or that hidden
tests passed.
```

**Coder, initial implementation**

```
# Task Introduction
{problem}

# Planner Plan
{planner_output}

# Your Role
You are a "Coder Agent". Independently check the plan, then solve the task.
No execution tool is available in this turn. Do not claim that you ran code or that hidden
tests passed.
Return one complete, self-contained Python program inside exactly one markdown ```python
code block.
The code block must contain only executable code. Do not include tests, examples, or
commentary inside it.
Follow the task's exact function signature or standard-input/standard-output contract.
```

**Tester, test generation**

```
# Task Introduction
{problem}

# Planner Plan
{planner_output}

# Coder Implementation
{coder_output}

# Public Example Execution
{public_execution}

# Your Role
You are a "Tester Agent". Design one high-value test that is likely to expose a concrete
defect in the current implementation.
No execution tool is available in this turn. Do not claim that you ran the code or that
hidden tests passed.
```

```
Return exactly one raw JSON object and no markdown fence or other text. Choose exactly one
mode.

For a concrete high-confidence test, return exactly these five string fields:
- "mode": "generated_test"
- "stdin": raw standard input, or an empty string for a function-call task.
- "driver": an empty string for a standard-input task, or minimal Python statements that
call the public function and print its return value.
- "expected_output": the exact stdout expected from the test.
- "rationale": a concise explanation of the implementation-specific bug or edge case
targeted.

If you cannot derive a reliable exact oracle, return exactly:
{"mode":"abstain","reason":"..."}
If the public examples pass and you find no concrete defect, return exactly:
{"mode":"no_issue","reason":"..."}

Populate exactly one of "stdin" and "driver"; never make both non-empty.
If the task says to read from stdin and write to stdout, "driver" must be exactly "". Do
not put a copy of the candidate, a checker, or print(expected_output) in "driver".
Before returning generated_test, silently verify the complete input grammar, including test
-case counts, dimensions, declared lengths, and public constraints. Silently verify that
expected_output agrees with rationale.
Derive the expected output from the public task specification, not from the Coder
implementation. Do not include hidden tests.
Do not reuse a public example merely to show that the implementation works. Use it only
when the current implementation failed that example.
```

**Tester, diagnosis of a failed public example**

```
# Task Introduction
{problem}

# Planner Plan
{planner_output}

# Coder Implementation
{coder_output}

# Trusted Public Example Execution
{public_execution}

# Your Role
You are a "Tester Agent" diagnosing a failure on examples already published in the task
statement.
The expected outputs above are trusted public evidence. Identify the concrete root cause in
 the current implementation and give a minimal repair direction.
Do not generate another test. Do not claim that hidden tests passed.
Return exactly one raw JSON object and no markdown fence or other text with exactly three
string fields:
{"mode":"diagnose_public_failure","diagnosis":"...","repair_hint":"..."}
```

**Coder, revision**

```
# Task Introduction
{problem}

# Planner Plan
{planner_output}

# Previous Coder Implementation
{coder_output}

# Public Example Execution
{public_execution}

# Tester Action
{tester_output}

# Deterministic Sandbox Execution Report
{tester_execution}

# Your Role
You are the "Coder Agent" revising the implementation after a Tester-authored test was
executed in a deterministic sandbox.
```

```
Treat the test and expected output as a hypothesis: independently check them against the
task, preserve correct parts, and fix only real defects.
The report states only what the candidate actually printed or whether it crashed; it does
not prove that the Tester oracle is correct.
Public example expected outputs are trusted because they were extracted from the task
statement and cross-checked against the dataset. A generated test remains an untrusted
hypothesis.
If the Tester abstained, reported no issue, supplied an invalid input, contradicted its own
 rationale, or a generated test matched, default to <keep/> unless you independently
identify a concrete defect.
No execution tool is available in this turn. Do not claim that you ran additional code or
that hidden tests passed.
Return one complete, self-contained Python program inside exactly one markdown ```python
code block, or return exactly <keep/> if the previous implementation should remain
unchanged.
The code block must contain only executable code. Do not include tests, examples, or
commentary inside it.
Follow the task's exact function signature or standard-input/standard-output contract.
```

### D.3 GAMES

Each player receives the environment's observation for its seat, followed by the instruction below. The legal-action guidance lists the currently legal actions and is recomputed by the environment before every move.

**Player (appended to the observation)**

```
{observation}
# Decision Protocol
Treat the observation above as authoritative. Use only information visible to your
player. Analyze the objective, legal actions, immediate threats or opportunities,
relevant opponent behavior or beliefs, and the consequences of your move.

# Canonical Action Contract
{action_contract}

# Current Legal-Action Guidance
{action_guidance}

Reason concisely in free-form prose. End on a new final line with exactly one legal
game action, including its square brackets, inside LaTeX \boxed{}. Output nothing
after the box. Only the boxed action is sent to the game or the other player.
```

**Action contracts**

```
Connect Four: Use [col N], where N is an integer from 0 through 6 and the selected column
is not full.

Liar's Dice: Use [Bid: Q, F], where Q is a positive integer and F is 1 through 6, or use [
Call]. Call is legal only when a current bid exists. A new bid must not decrease Q or F and
 must strictly increase at least one of them. After a New round message, the previous bid
is no longer current: make a Bid rather than a Call.

Colonel Blotto: Use [A<units> B<units> C<units>] with non-negative integers whose sum is
exactly 20, for example \boxed{[A7 B7 C6]}. Ignore any conflicting example whose
allocations do not sum to 20.

Breakthrough: Use [FROMTO], for example [a2a3], with files a-f and ranks 1-6. Player 0/
White owns W and moves exactly one rank upward; Player 1/Black owns B and moves exactly one
 rank downward (for example [a5a4] initially). Move only your own piece, straight into an
empty square or diagonally to capture an opponent piece.
```