MASHAP-LLM: CONTRIBUTION-AWARE POLICY OPTIMIZATION FOR MULTI-AGENT LLM POST-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent large language model (LLM) systems face a credit assignment problem during training. Shared rewards give every agent the same feedback regardless of its contribution, so policy updates have no explicit signal of which agent's behavior to reinforce. Measuring each agent's contribution by rerunning the task with subsets of agents, called coalition rollouts, is costly. We propose MASHAP-LLM, a multi-agent LLM post-training algorithm that rewards each agent according to its contribution and needs coalition rollouts only early in training. MASHAP-LLM builds on proximal policy optimization (PPO) and assigns rewards using Shapley values. At its core is LLM-SHAP, a learned component that reads the task and the agents' trajectories and estimates the coalition values used to compute these contributions. LLM-SHAP is trained on coalition rollouts scored by a verifier during an early fitting phase and then frozen. After freezing, LLM-SHAP scores each coalition from the full-team trajectory with the outputs of absent agents masked, so no smaller team is generated or verified. MASHAP-LLM turns the resulting Shapley credits into per-agent rewards through a rescaling that preserves the total team reward. We evaluate 3B models from the Qwen2.5 family on three math benchmarks and three coding benchmarks. Against the strongest baseline in each of nine settings, MASHAP-LLM improves the average score by 2.51 percentage points with two agents and 4.36 points with three agents.

1 INTRODUCTION

Multi-agent systems (MAS) organize large language models (LLMs) into specialized roles that communicate to solve shared tasks. CAMEL and AutoGen illustrate how role-based interaction supports collaborative problem solving (Li et al., 2023, Wu et al., 2024). By separating reasoning, solution generation, and review, agents can build on intermediate results and revise one another's outputs. Multi-agent collaboration offers a promising approach to mathematical reasoning and programming (Zhao et al., 2026).

Reinforcement learning (RL) improves mathematical reasoning and code generation through task feedback (Shao et al., 2024, Le et al., 2022). MARFT, MAPoRL, and Stronger-MAS extend RL to collaborating LLMs (Liao et al., 2025, Park et al., 2025, Zhao et al., 2026). A central issue is how to translate team outcomes into learning signals for individual agents. Sharing a team reward leaves individual attribution implicit, a longstanding challenge in cooperative RL (Foerster et al., 2018). To provide more informative feedback, C3 compares context-matched alternatives (Chen et al., 2026), CCPO constructs agent-removal counterfactuals (Li et al., 2026b), and SHARP incorporates Shapley-inspired marginal credit into a decomposed reward (Li et al., 2026a). These approaches leave open how coalition information can support contribution-aware updates without repeatedly executing alternative teams throughout training.

Addressing this problem involves two coupled challenges. First, **credit assignment must account for interdependent agent outputs.** In sequential collaboration, upstream responses shape downstream outputs, while the final outcome alone does not identify each role's contribution. Simply removing a message from a completed transcript leaves downstream answers that were generated using that message. Reference outcomes for a reduced team therefore require re-executing participating agents

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.

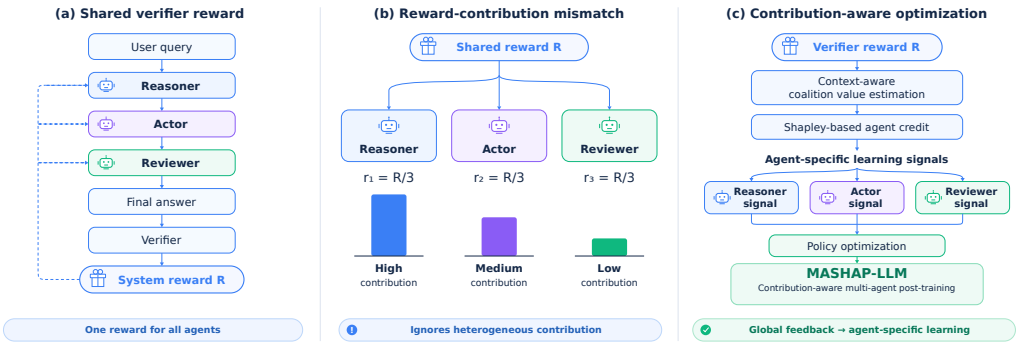


Figure 1: MASHAP-LLM converts team feedback into contribution-aware updates. Early coalition executions supervise value estimation, and frozen prediction supports subsequent allocation. Shapley credits are rescaled to preserve the team reward. The contribution ordering is schematic and task-dependent.

under the changed context. Second, **coalition-based credit must remain computationally practical during training.** Shapley allocation considers marginal contributions across subsets of the team (Shapley, 1953), but obtaining these coalition values through repeated generation and verification is expensive. Learned attribution can amortize repeated computation (Jethani et al., 2022), motivating an estimator that reuses coalition supervision as the collaborating policies continue to improve.

To address these challenges, we introduce MASHAP-LLM, a contribution-aware post-training algorithm that separates the acquisition of coalition supervision from its repeated use in policy optimization. We first execute agent coalitions to obtain verifier-labeled trajectories and train a contextual coalition-value estimator, LLM-SHAP. Its predictions are converted into Shapley credits and rescaled to preserve the observed team reward, yielding agent-specific rewards for PPO updates (Schulman et al., 2017). After an early fitting phase, we freeze the estimator and predict coalition values from masked full-team trajectories. The masked trajectories are inputs to the estimator, and the verifier does not score them. The frozen phase requires estimator inference and full-team feedback, but no strict-subcoalition generation or verification. Figure 1 illustrates how team feedback becomes contribution-aware learning signals.

We evaluate MASHAP-LLM on mathematical reasoning and programming with Qwen2.5-family 3B models. Across nine equally weighted settings, MASHAP-LLM exceeds the strongest baseline in each setting by 2.51 points on average with two agents and 4.36 points with three agents. Freezing the estimator reduces full-run GPU-hours by 10.24% on mathematics and 10.83% on code, compared with refreshing it once during training. The frozen variant stays within the prespecified noninferiority margins of 1.0 point on MATH500 and 0.5 point on Codeforces. Replacing task-aligned coalition labels with task-shuffled labels lowers performance by 2.08 points on MATH500 and 0.63 points on Codeforces.

Our contributions are as follows.

- **Contribution-aware policy optimization.** We develop MASHAP-LLM, which connects coalition-value predictions to agent-specific PPO updates through Shapley allocation and reward rescaling that preserves the team reward.
- **Early-fit, frozen-use coalition estimation.** We introduce LLM-SHAP and a training procedure that learns from early coalition executions, then supports reward allocation without further strict-subcoalition generation or verification.
- **Empirical gains and component analysis.** We demonstrate gains over shared-reward and credit-assignment baselines on reasoning and programming tasks, with resource comparisons and ablations assessing coalition information, credit aggregation, and estimator freezing.

2 Related Work

Multi-Agent LLM Post-Training. Role-based frameworks coordinate LLMs through prompts and message exchange (Li et al., 2023, Wu et al., 2024). MARFT develops multi-agent reinforcement fine-tuning (Liao et al., 2025). MAPoRL jointly post-trains collaborators (Park et al., 2025). MALT derives role-specific training data from expanded trajectories (Motwani et al., 2025), and MAGRPO extends group-relative optimization to multi-agent interaction (Liu et al., 2026). Stronger-MAS combines agent- and turn-wise grouping, tree sampling, and team and local rewards (Zhao et al., 2026). Our focus is learning coalition values to allocate team feedback for individual policy updates.

Counterfactual and Shapley-Based Credit Assignment. COMA uses counterfactual advantages (Foerster et al., 2018), while Shapley Counterfactual Credits and SHAQ incorporate coalition attribution into cooperative learning (Li et al., 2021, Wang et al., 2022). For LLM collaboration, C3 compares context-matched actions (Chen et al., 2026), CCPO uses agent-removal counterfactuals (Li et al., 2026b), and SHARP combines Shapley marginal credit with other reward components and grouped normalization (Li et al., 2026a). Other work derives agent- and message-level evaluation signals (Yang et al., 2025b). ShapE-GRPO allocates credit among candidate responses (Ai et al., 2026). MASHAP-LLM instead learns from verifier-scored coalition executions and uses predicted coalition values during policy training. Appendix A compares these mechanisms.

Learned Coalition Values and Amortized Attribution. Value-function learning supports on-manifold attribution (Frye et al., 2021). FastSHAP amortizes Shapley estimation (Jethani et al., 2022), and ViT-Shapley combines masking with learned attribution (Covert et al., 2023). Our reference targets come from executing LLM coalitions, and the resulting allocations are policy-training rewards. Early fitting on re-rollout contexts followed by frozen prediction from masked trajectories introduces an input change. We therefore assess prediction fidelity and downstream training performance separately.

3 Preliminaries and Problem Setup

MAS Setting. We model a team $\mathcal{N} = \{1, \dots, n\}$ of role-specialized LLM agents as a finite-horizon, turn-based cooperative Markov game (Littman, 1994) with a shared terminal reward. For a task $q \sim \mathcal{Q}$, agents act once each in a fixed order. Before agent i acts, the state is $s_i = (q, i, a_1, \dots, a_{i-1})$. Agent i receives its role prompt and context $c_i = (q, a_1, \dots, a_{i-1})$, and produces a complete text response a_i using policy π_{θ_i}. Appending this response gives the deterministic transition $s_{i+1} = (q, i+1, a_1, \dots, a_i)$. Each response is generated token by token, but the complete response is the action to which we assign credit. One episode produces the trajectory $\boldsymbol{a} = (a_1, \dots, a_n)$ for one task, and b indexes independently sampled tasks. Agents use role-specific low-rank adaptation (LoRA) parameters on a common backbone initialization (Appendix E).

MAS Workflow. Our two-agent workflows are Reasoner $\rightarrow$ Actor for mathematical reasoning and Reasoner $\rightarrow$ Coder for code generation. The three-agent workflows append a Reviewer that checks the preceding solution, corrects it if needed, and submits the final answer. In the two-agent setting, the Actor or Coder submits the final answer. The Reviewer is an LLM agent, distinct from the external verifier that scores the completed task. Role prompts are provided in Appendix F.

System-level Feedback and Objective. An external verifier evaluates the completed trajectory and returns a single bounded reward $R_b = \mathrm{Verifier}(q_b, \boldsymbol{a}_b) \in [0, R_{\max}]$. The training objective is to improve the expected verifier score under the task distribution and a specified decoder D,

$$J^D_{\mathrm{sys}}(\boldsymbol{\theta}) = \mathbb{E}_{q \sim \mathcal{Q},\, \boldsymbol{a} \sim P^D_{\boldsymbol{\theta}}(\cdot|q)}[\mathrm{Verifier}(q, \boldsymbol{a})]. \tag{1}$$

Here $P^D_{\boldsymbol{\theta}}$ is the trajectory distribution induced by the sequential workflow and decoder. MASHAP-LLM uses greedy decoding with one full-team trajectory per task during policy training. Although R_b evaluates the team outcome, it does not specify how the individual roles should be reinforced. Our method constructs role-specific training signals from this shared feedback.

Coalition Values and Shapley Credit. The Markov game describes sequential interaction, while a cooperative coalition game defines credit allocation. A coalition $C \subseteq \mathcal{N}$ specifies which roles participate in a task. For a fixed behavior-policy snapshot, let $\boldsymbol{a}^C$ denote the trajectory generated

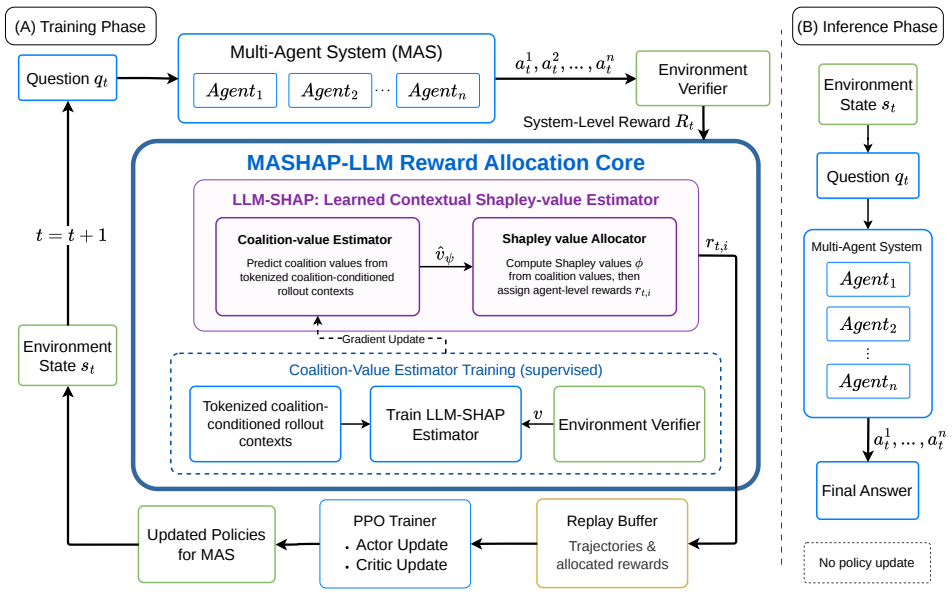


Figure 2: MASHAP-LLM training (left) and task-time inference (right). Blocks B–C of Algorithm 1 separate value estimation and credit allocation. Frozen estimation uses masked inputs.

under that participation condition. The verifier defines a reference coalition value

$$v_{q,\boldsymbol{a}}(C) = v^{\text{env}}_{q,\boldsymbol{a}}(C) = \text{Verifier}(q, \boldsymbol{a}^C). \tag{2}$$

Because roles interact sequentially, removing a role can change subsequent responses. Section 4.1 specifies the corresponding re-rollout protocol. Given any coalition-value function v, the Shapley value (Shapley, 1953) assigns each role its weighted average marginal contribution,

$$\phi_i(v) = \sum_{C \subseteq \mathcal{N} \setminus \{i\}} w_{|C|} \left[v(C \cup \{i\}) - v(C) \right], \qquad w_k = \frac{k!(n-k-1)!}{n!}. \tag{3}$$

Computing these credits requires coalition values in addition to the observed full-team reward. MASHAP-LLM obtains them using a learned component, LLM-SHAP, and converts the resulting credits into role-specific policy-update targets. We distinguish the reference game v^{env} from the learned value function $\hat{v}_\psi$. We test fidelity to v^{env} and usefulness as a training signal separately.

4 The MASHAP-LLM Post-Training Algorithm

MASHAP-LLM combines team rollout, coalition-value prediction, credit allocation, and policy update (Figure 2). **LLM-SHAP is its learned coalition-value estimation component.** Its predicted value table feeds deterministic Shapley allocation and reward rescaling, followed by PPO updates. Algorithm 1 marks these responsibilities.

Early coalition re-rollouts supervise the estimator. Frozen prediction then removes strict-subcoalition environment interactions. Learned values can also change credit under binary feedback. Their optimization benefit and its dependence on rescaling remain empirical questions, examined in Section 5 and Appendix I.1.

4.1 Coalition Supervision and LLM-SHAP Value Estimation

Reference labels from coalition re-rollouts. For coalition C, each excluded role outputs a fixed absence message. Participating roles generate in order under the same behavior-policy snapshot and greedy decoder, so downstream responses are regenerated. CoalTrain constructs $(C, x_C^{\text{train}}, y_C)$ with $x_C^{\text{train}} = \text{BuildCtx}(q_b, \boldsymbol{a}_b^C, C)$ and $y_C = \text{Verifier}(q_b, \boldsymbol{a}_b^C)$. The grand coalition reuses R_b. Coalitions without an answer-capable role receive zero under the absence convention (Appendix D.3).

Algorithm 1 MASHAP-LLM post-training. Block B is the LLM-SHAP component.

Input: Role policies $\{\pi_{\theta_i}\}$, verifier Verifier, LLM-SHAP estimator $\hat{v}_\psi$, critic V_ω
Input: Batch size B, PPO settings, normalization floor τ_{floor}, freezing rule λ_{stop}
Output: Updated role policies, estimator, and critic

frozen $\leftarrow$ False
for each policy update **do**
  $\boldsymbol{\theta}_{\text{old}} \leftarrow \boldsymbol{\theta}$; initialize rollout buffer $\mathcal{B} \leftarrow \emptyset$
  for each task q_b in a batch of B tasks sampled from $\mathcal{Q}$ **do**
  A. Team rollout and system feedback
    $(\boldsymbol{a}_b, \Omega_b) \leftarrow \textsc{GreedyTeamRollout}(q_b, \boldsymbol{\theta}_{\text{old}}, V_\omega)$
    $R_b \leftarrow \text{Verifier}(q_b, \boldsymbol{a}_b)$
  B. LLM-SHAP component: contexts, prediction, and fitting
    if not frozen **then**
      $\mathcal{D}_b^{\text{coal}} \leftarrow \textsc{CoalTrain}(q_b, \boldsymbol{a}_b, R_b, \text{Verifier}, \boldsymbol{\theta}_{\text{old}})$
    else
      $\mathcal{D}_b^{\text{coal}} \leftarrow \textsc{CoalInfer}(q_b, \boldsymbol{a}_b)$ ▷ masked contexts only
    $\widehat{\mathcal{V}}_b \leftarrow \textsc{InferV}(\mathcal{D}_b^{\text{coal}}, \hat{v}_\psi)$ ▷ predicted values
    if not frozen **then**
      $\psi \leftarrow \textsc{FitV}(\mathcal{D}_b^{\text{coal}}, \psi)$ ▷ supervised estimator update
      $(\psi, \text{frozen}) \leftarrow \textsc{FreezeCheck}(\psi; \lambda_{\text{stop}})$
  C. Shapley allocation and reward rescaling
    $\hat{\boldsymbol{\phi}}_b \leftarrow \textsc{Shapley}(\widehat{\mathcal{V}}_b)$ ▷ Eq. (3), exact or sampled
    $\boldsymbol{r}_b \leftarrow \textsc{Rescale}(\hat{\boldsymbol{\phi}}_b, R_b, \tau_{\text{floor}})$ ▷ Eq. (4)
    Store $(\Omega_b, \boldsymbol{r}_b)$ in $\mathcal{B}$
  D. Role-specific policy and critic updates
  Compute fixed advantages $A_{b,i} = r_{b,i} - V_{b,i}$ for $\mathcal{B}$
  Update role policies and critic using Eq. (6). Keep allocated rewards fixed

The learned component and its outputs. LLM-SHAP fits an LLM backbone and scalar value head to verifier labels by squared error. InferV returns $\widehat{\mathcal{V}}_b(C) = \hat{v}_\psi(x_C)$ for every coalition, including the grand coalition. Predictions supply the allocation table in both phases. Labels are never substituted. The observed R_b is used separately for rescaling. Appendix E.8 gives estimator settings.

Freezing and masked-context prediction. A held-out stopping rule freezes the best validation checkpoint (Appendix E.8.1). CoalInfer then builds $x_C^{\text{infer}} = \textsc{MaskCtx}(q_b, \boldsymbol{a}_b^{\mathcal{N}}, m_C)$, where $m_C \in \{0,1\}^n$ selects coalition members, hiding non-members without regenerating retained responses. Retained responses were generated with the full-team context, so inference requires transfer from re-rollout to masked contexts. Appendix I.2 evaluates fidelity against re-rollout references. Direct verification of masked transcripts is a separate ablation (Appendix D.4).

4.2 From Coalition Values to Role-Specific Credit

The Shapley operator in Eq. (3) converts $\widehat{\mathcal{V}}_b$ into raw credits $\hat{\boldsymbol{\phi}}_b$. Their sum is determined by the predicted endpoint values and need not equal the observed reward R_b. We rescale them to preserve the realized reward budget, using equal allocation when the denominator is too small or its sign is inconsistent,

$$r_{b,i}^{\text{LLM-SHAP}} = \begin{cases} \dfrac{R_b \hat{\phi}_{b,i}}{\hat{\Sigma}_b}, & |\hat{\Sigma}_b| \geq \tau_{\text{floor}} \text{ and } \operatorname{sign}(\hat{\Sigma}_b) = \operatorname{sign}(R_b), \\ R_b/n, & \text{otherwise}, \end{cases} \qquad \hat{\Sigma}_b = \sum_j \hat{\phi}_{b,j}. \tag{4}$$

The superscript identifies rewards obtained using LLM-SHAP predictions. The deterministic allocation and rescaling are distinct from the learned component. Both branches satisfy $\sum_i r_{b,i} = R_b$. Appendix C gives the budget identities and conditional error analysis. Appendix I.5 reports allocation diagnostics. Appendix D gives the combined Alloc subroutine for block C.

4.3 Role-Specific Advantages and Policy Updates

The rollout buffer $\Omega_b = \{c_{b,i}, a_{b,i}, \ell_{b,i}, V_{b,i}\}_{i=1}^n$ stores the context, response, old log-likelihood, and baseline value of each role. The critic V_ω conditions on the role's input context and regresses to its

Table 1: Mathematical reasoning Pass@1 (%): mean ± sample SD over five seeds (divisor 4). Training domains head each block. Best results are bold. SFT: supervised fine-tuning, direct-inference reference.

SFT backbone	**MATH500**	**CMATH**	**GSM8K**
$n = 2$	44.80	79.96	74.75
$n = 3$	40.40	79.87	75.97

Method	**MATH training**			**CMATH training**		
	MATH500	CMATH	GSM8K	MATH500	CMATH	GSM8K
			Two agents ($n = 2$)			
MARFT-A	46.08 ± 1.29	80.26 ± 0.88	76.13 ± 0.46	46.80 ± 0.68	80.22 ± 0.96	74.97 ± 0.35
SHARP	46.52 ± 0.58	80.35 ± 1.03	76.42 ± 0.76	46.52 ± 0.33	80.11 ± 0.62	74.60 ± 0.82
CCPO	47.60 ± 0.65	82.40 ± 1.02	78.35 ± 0.88	48.92 ± 0.44	82.37 ± 0.71	77.71 ± 0.78
C3	48.60 ± 0.81	82.28 ± 1.05	78.51 ± 0.88	49.40 ± 0.35	82.44 ± 0.57	77.60 ± 0.93
MASHAP-LLM	$\mathbf{52.28 \pm 0.48}$	$\mathbf{84.79 \pm 0.87}$	$\mathbf{80.61 \pm 0.73}$	$\mathbf{52.60 \pm 0.68}$	$\mathbf{84.15 \pm 0.72}$	$\mathbf{81.15 \pm 0.91}$
			Three agents ($n = 3$)			
MARFT-A	46.00 ± 0.99	80.71 ± 1.21	77.60 ± 0.79	46.80 ± 0.79	81.37 ± 1.35	76.50 ± 0.53
SHARP	46.12 ± 1.25	80.15 ± 1.23	77.15 ± 0.60	46.52 ± 0.44	81.17 ± 1.14	76.06 ± 0.24
CCPO	47.72 ± 0.72	80.51 ± 1.32	77.57 ± 1.52	45.92 ± 0.23	82.44 ± 0.84	77.85 ± 0.75
C3	48.12 ± 1.35	81.02 ± 1.13	78.77 ± 0.80	47.64 ± 0.67	83.46 ± 1.10	78.05 ± 0.56
MASHAP-LLM	$\mathbf{54.44 \pm 0.95}$	$\mathbf{85.63 \pm 1.02}$	$\mathbf{82.46 \pm 0.78}$	$\mathbf{54.80 \pm 0.45}$	$\mathbf{86.12 \pm 1.31}$	$\mathbf{82.49 \pm 0.83}$

allocated terminal credit. With one complete task per episode, we use the baseline-subtracted training signal

$$A_{b,i} = r_{b,i}^{\text{LLM-SHAP}} - V_\omega(c_{b,i}). \tag{5}$$

The credit becomes available after all roles finish. No return is bootstrapped across independent tasks. Each role updates its own LoRA parameters using its responses and fixed advantages. We maximize the PPO-Clip surrogate (Schulman et al., 2017)

$$\begin{aligned} \mathcal{L}_i^{\text{PPO}} = \mathbb{E}_{\mathcal{B}}\Big[\min\Big(\rho_{b,i}A_{b,i}, \text{clip}_{\varepsilon_{\text{clip}}}(\rho_{b,i})A_{b,i}\Big)\Big] \\ - c_v\mathbb{E}_{\mathcal{B}}\Big[\big(V_\omega(c_{b,i}) - r_{b,i}^{\text{LLM-SHAP}}\big)^2\Big], \end{aligned} \tag{6}$$

where $\rho_{b,i} = \pi_{\theta_i}(a_{b,i} \mid c_{b,i})/\pi_{\theta_{i,\text{old}}}(a_{b,i} \mid c_{b,i})$ and $\varepsilon_{\text{clip}} = 0.2$. The coefficient c_v weights the critic's squared error. The expectation is over the fixed rollout batch. Actor differentiation holds the allocated rewards and advantages fixed. Because these batches are generated greedily, we use the clipped expression as an empirical training surrogate. The likelihood ratio is not an importance weight against the deterministic behavior policy. We also do not claim that $A_{b,i}$ is an unbiased advantage of the system return. Appendix C.3 states the corresponding fixed-data derivative and its interpretation.

4.4 Training Schedule and Computational Cost

Algorithm 1 holds the behavior policy fixed within each update. During fitting, prediction precedes estimator optimization and the freezing check. Allocated rewards remain fixed. After freezing, block B only constructs contexts and predicts values. Appendix D details the subroutines and schedule.

For $n \leq 3$, allocation enumerates 2^n coalitions. Larger teams use K shared permutations, averaging $\mathcal{O}(Kn)$ marginal terms (Appendix I.3). Fitting uses coalition re-rollouts. Frozen allocation uses masked contexts, one full-team rollout, and its verifier reward. Estimator inference remains necessary, and sampling need not save work at four or five agents. Section 5.4 reports the end-to-end cost.

5 Experiments

We assess task performance (Tables 1–2), component choices, resource budgets, and larger teams. Appendix K provides paired replication, complete budget results, and statistical protocols.

5.1 Experimental Setup

Tasks and evaluation. Math models train on MATH (Hendrycks et al., 2021) or CMATH (Wei et al., 2023) and are tested on MATH500, CMATH, and GSM8K (Cobbe et al., 2021), covering cross-dataset

Table 2: Code generation: mean unit-test pass rate (%), five-run mean ± sample SD where shown. Trained methods share Codeforces checkpoints across in-distribution (ID) and out-of-distribution (OOD) columns. SFT is a direct-inference reference.

Method	Two agents ($n = 2$)			Three agents ($n = 3$)		
	Codeforces (ID)	MBPP-test (OOD)	MBPP+ (OOD)	Codeforces (ID)	MBPP-test (OOD)	MBPP+ (OOD)
SFT	4.33	5.31	5.24	1.83	12.15	11.79
MARFT-A	4.52 ± 0.25	5.78 ± 0.80	5.62 ± 0.99	4.46 ± 0.34	12.70 ± 1.41	12.35 ± 1.78
SHARP	4.51 ± 0.31	6.15 ± 0.89	6.02 ± 0.80	4.41 ± 0.23	13.40 ± 1.58	13.05 ± 1.45
CCPO	4.60 ± 0.33	6.40 ± 0.91	6.58 ± 0.68	4.74 ± 0.28	14.30 ± 1.64	14.19 ± 1.21
C3	4.74 ± 0.41	6.92 ± 0.95	6.85 ± 0.74	4.83 ± 0.22	15.05 ± 1.72	14.85 ± 1.34
MASHAP-LLM	**5.79 ± 0.24**	**9.65 ± 0.83**	**9.14 ± 0.71**	**6.47 ± 0.37**	**19.78 ± 1.48**	**18.84 ± 1.26**

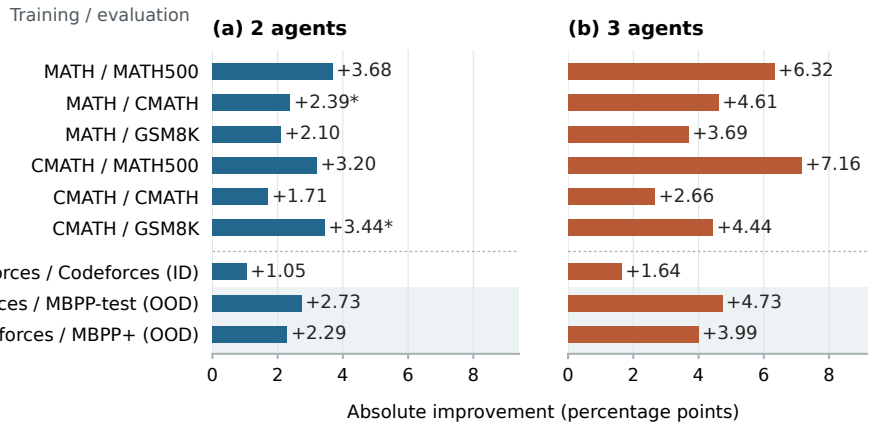


Figure 3: Gains over the strongest baseline in each of six math and three code settings (Tables 1–2). Gains use the table means. Stars: CCPO, otherwise C3.

and English–Chinese transfer. Codeforces-trained models are tested on held-out Codeforces and, unchanged, MBPP-test/MBPP+ (Appendix E.3.1). Math reports greedy Pass@1. Code reports mean unit-test pass rate.

Models and training. Math uses Qwen2.5-Coder-3B-Instruct (Hui et al., 2024). Code uses Qwen2.5-3B-Instruct (Yang et al., 2025a), with role-specific LoRA adapters (Hu et al., 2022). All methods generate one greedy team trajectory per test problem. Main comparisons use five seeds and final-update checkpoints (Appendix E). Paired controls use separate seeds and family-wise inference (Appendix K.1). Resource protocols appear in Appendix K.3.

Baselines and comparison protocol. We compare frozen SFT, equal-allocation MARFT-A (Liao et al., 2025), and C3, CCPO, and SHARP (Chen et al., 2026, Li et al., 2026b;a). Methods retain their sampling and update mechanisms under common tasks, models, and evaluation (Appendix E.1). Resource comparisons also include SEPO (Li et al., 2026b), AT-GRPO (Zhao et al., 2026), and Shared-GRPO (Table 4).

5.2 Main Results

Tables 1–2 contain six mathematical train–test settings and three code evaluations. Across these nine settings, MASHAP-LLM exceeds the strongest baseline in each setting by an average of 2.51 points with two agents and 4.36 points with three agents (Figure 3). The strongest baseline is C3 in 16 of the 18 settings and CCPO in the other two.

Paired replication. Fixed-update replication uses 24 paired seeds per mathematical training domain and 12 for code (Table 29), separately from the five-seed main results. All 18 reported intervals exclude zero under the prespecified 194-comparison correction. Descriptive averages are 2.48/4.33 points over the named comparators. Repeated benchmarks and overlapping MBPP tasks are dependent.

Transfer across datasets. Gains extend to held-out mathematical and code benchmarks. Each run uses the same final checkpoint across its test benchmarks. MATH and CMATH training differ in data size and update intensity. Their results do not establish data efficiency.

5.3 Ablation Study

Variant	MATH500	CMATH	GSM8K	Code
Dual-Agent *($n = 2$)*				
Ours (L+CF-train)	**52.28±0.48**	**84.79±0.87**	**80.61±0.73**	**5.79±0.24**
Verifier (CF)	48.36±0.17	81.26±1.03	77.33±0.31	4.64±0.53
Static (No L/CF)	47.24±0.54	75.43±1.55	71.90±0.11	3.84±0.46
Equal allocation (reference)	*46.08±1.29*	*80.26±0.88*	*76.13±0.46*	*4.52±0.25*
Triple-Agent *($n = 3$)*				
Ours (L+CF-train)	**54.44±0.95**	**85.63±1.02**	**82.46±0.78**	**6.47±0.37**
Verifier (CF)	49.60±1.05	82.79±1.05	79.45±0.62	5.47±0.16
Static (No L/CF)	47.20±0.79	77.19±1.20	73.01±1.00	4.76±0.40
Role prior (fixed shares)	47.48±0.11	77.56±1.10	72.45±0.50	4.51±0.20
Equal allocation (reference)	*46.00±0.99*	*80.71±1.21*	*77.60±0.79*	*4.46±0.34*

Table 3: Coalition-value ablations, mean ± SD. L: learned estimator, CF: coalition re-rollout. Role prior uses fixed shares. Equal is the reference. Full protocol: Appendix B.

Table 3 compares learned values, verifier-scored re-rollouts, and static masking with Shapley allocation. Learned values also use the rescaling of Section 4.2. Direct verifier Shapley already sums to the team reward (Appendix D.3).

Coalition information and aggregation. The pipeline comparison favors learned allocation (Table 3). Paired component controls isolate task alignment, label information, and aggregation (Tables 34–36). Full-minus-task-shuffled differences are $2.08/0.63$ points on MATH500/Codeforces. Full-minus-LOO differences are $1.36/0.49$. Their reported corrected intervals exclude zero. The oracle comparison uses learned-rerolled versus oracle-rerolled inputs, isolating value estimation under matched contexts.

Rescaling and contextual allocation. The fixed role prior does not reproduce full-method performance. Paired controls favor multiplicative rescaling over additive centering by $1.00/0.44$ points at the primary endpoints. Mixed raw-credit results do not establish uniform superiority of rescaling.

5.4 Efficiency and Scalability

Resource-matched performance. Table 4 compares three-agent systems ($n = 3$) against six baselines, with shared domain-specific backbones, role configurations, training data, and evaluation protocols. Runs use the final checkpoint permitted by each GPU-hour or generated-output-token budget. Mean margins over the strongest competitor (SEPO, or C3 for Codeforces under the token budget) are $1.79/1.73$ points on MATH500 and $0.64/0.63$ on Codeforces under the respective budgets. All six benchmarks show positive mean margins (Appendix K.3).

End-to-end training cost. Learned and verifier-induced allocation take $1.473\times$ and $1.563\times$ single-agent PPO time (Table 13). Frozen estimation saves $10.24\%/10.83\%$ full-run GPU-hours in math/code versus refresh, with positive cost-family intervals (Table 38). Frozen and masked variants satisfy prespecified $1.0/0.5$-point noninferiority margins on MATH500/Codeforces (Table 37).

Larger teams. Four agents use Decomposer, Planner, Executor, and Verifier. Five add Reflector. Table 5 reports five-seed performance with 20 shared orders. Separate paired replication yields positive corrected margins over C3 (Table 30). Team size changes with roles. OOD code tests cover $n = 2, 3$ only.

Method	GPU-hour envelope		Output-token envelope	
	MATH500	Codeforces	MATH500	Codeforces
Reported resource range	2379.0–2386.2 h	3170.5–3182.7 h	5.973–5.978 B	4.475–4.484 B
MASHAP	$\mathbf{54.36 \pm 1.02}$	$\mathbf{6.50 \pm 0.36}$	$\mathbf{54.56 \pm 0.98}$	$\mathbf{6.58 \pm 0.34}$
C3	52.34 ± 1.27	5.72 ± 0.34	52.27 ± 1.12	5.95 ± 0.38
CCPO	51.59 ± 0.96	5.57 ± 0.48	51.98 ± 1.14	5.71 ± 0.40
SEPO	52.57 ± 1.10	5.86 ± 0.35	52.83 ± 1.04	5.87 ± 0.33
SHARP	51.45 ± 0.98	5.50 ± 0.37	51.57 ± 1.18	5.44 ± 0.37
AT-GRPO	52.47 ± 1.10	5.80 ± 0.38	52.78 ± 1.07	5.86 ± 0.33
Shared-GRPO	51.19 ± 1.04	5.31 ± 0.30	51.46 ± 1.30	5.46 ± 0.41

Table 4: Three-agent resource-matched performance ($n = 3$), mean $\pm$ sample SD (%). MATH500: MATH training, Pass@1. Code: Codeforces training, mean unit-test pass rate. Separate budgets constrain GPU-hours or generated output tokens. Ranges span the seven methods' reported GPU-hours (h) or billion output tokens (B). Complete scores: Tables 31–32. Resource accounting: Table 33.

Table 5: Four/five-agent performance, five-seed mean $\pm$ SD, $K = 20$ shared orders. Math/code use MATH/Codeforces training.

Test	Four agents		Five agents	
	Ours	C3	Ours	C3
MATH500	$\mathbf{55.1 \pm 0.7}$	51.4 ± 0.9	$\mathbf{54.3 \pm 0.8}$	50.6 ± 1.0
CMATH	$\mathbf{85.9 \pm 0.8}$	81.2 ± 1.0	$\mathbf{85.5 \pm 0.9}$	80.9 ± 1.1
GSM8K	$\mathbf{82.8 \pm 0.7}$	78.1 ± 0.9	$\mathbf{82.4 \pm 0.8}$	77.8 ± 1.0
Code	$\mathbf{6.55 \pm 0.31}$	4.95 ± 0.30	$\mathbf{6.42 \pm 0.34}$	4.89 ± 0.32

Table 6: Frozen-estimator drift (a) and fitting-label-noise robustness (b). Protocols and held-out sets: Appendix B.

(a) Post-freeze drift

Trajectories	Value MAE$_v$	Credit MAE$_\phi$
During training	0.051387	0.026442
Final policies	0.053687	0.027617

(b) Corrupted fitting labels

Label flips	Credit-rank agreement	MATH500
0%	93%	54.44 ± 0.95
5%	91%	54.08 ± 0.88
10%	88%	53.40 ± 0.94
20%	82%	51.80 ± 1.03

5.5 Post-Freeze Reliability and Verifier Robustness

Prediction after freezing. LLM-SHAP freezes within 10–15% of updates. Relative to coalition re-rollouts, value/credit MAEs rise by under 0.003 from training to final policies (Table 6a, Appendix I.2).

Noise in coalition supervision. Strict-subcoalition labels are symmetrically flipped by $0/5/10/20\%$. Grand-coalition rewards and evaluation stay fixed (Table 6b). At 10%, MATH500 drops 1.04 points. At 20%, 51.80 exceeds C3 (48.12), but C3 used clean labels, so this is not a matched comparison.

6 Conclusion

MASHAP-LLM rewards each agent with its Shapley credit, computed from coalition values that LLM-SHAP predicts and rescaled so that the rewards sum to the team reward. Across nine math and code settings, MASHAP-LLM exceeds the strongest baseline by an average of 2.51 points with two agents and 4.36 points with three agents. Freezing LLM-SHAP after early fitting reduces full-run GPU-hours by about 10% within the prespecified noninferiority margins.

7 Limitations

All experiments use Qwen2.5 models at the 3B scale with fixed roles, greedy decoding, and a programmatic verifier. The frozen estimator must transfer from re-rollout contexts to masked contexts while the policies change. On a failed task, every role receives zero reward, so the method cannot assign blame among roles. Appendix J gives details.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript writing and language polishing. The authors are responsible for the final manuscript, including all claims, results, and citations.

ETHICS STATEMENT

All datasets used in this work are publicly released research benchmarks. No personally identifiable information is collected or used. The proposed reward-allocation mechanism changes training-time credit assignment and does not introduce new user-facing capabilities beyond the underlying LLM systems. Standard misuse considerations for mathematical reasoning and code-generation models still apply.

REFERENCES

Rui Ai, Yu Pan, David Simchi-Levi, and Chonghuan Wang. ShapE-GRPO: Shapley-enhanced reward allocation for multi-candidate LLM training. *arXiv preprint arXiv:2603.29871*, 2026.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Yanjun Chen, Yirong Sun, Hanlin Wang, Xinming Zhang, Xiaoyu Shen, Wenjie Li, and Wei Zhang. Contextual counterfactual credit assignment for multi-agent reinforcement learning in LLM collaboration. *arXiv preprint arXiv:2603.06859v1*, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Ian Covert, Chanwoo Kim, and Su-In Lee. Learning to estimate Shapley values with vision transformers. In *International Conference on Learning Representations*, 2023.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Christopher Frye, Damien de Mijolla, Tom Begley, Laurence Cowton, Megan Stanley, and Ilya Feige. Shapley explainability on the data manifold. In *International Conference on Learning Representations*, 2021.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Neil Jethani, Mukund Sudarshan, Ian C. Covert, Su-In Lee, and Rajesh Ranganath. FastSHAP: Real-time Shapley value estimation. In *International Conference on Learning Representations*, 2022.

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. In *Advances in Neural Information Processing Systems*, volume 35, pp. 21314–21328, 2022.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36, pp. 51991–52008, 2023.

Jiahui Li, Kun Kuang, Baoxiang Wang, Furui Liu, Long Chen, Fei Wu, and Jun Xiao. Shapley counterfactual credits for multi-agent reinforcement learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 934–942, 2021.

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

Zhongyi Li, Wan Tian, Jinju Chen, Huiming Zhang, Yang Liu, Yikun Ban, and Fuzhen Zhuang. Counterfactual credit policy optimization for multi-agent collaboration. *arXiv preprint arXiv:2603.21563*, 2026b.

Junwei Liao, Muning Wen, Jun Wang, and Weinan Zhang. MARFT: Multi-agent reinforcement fine-tuning. *arXiv preprint arXiv:2504.16129*, 2025.

Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pp. 157–163. Morgan Kaufmann, 1994. doi: 10.1016/B978-1-55860-335-6.50027-1.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, pp. 21558–21572, 2023.

Shuo Liu, Zeyu Liang, Xueguang Lyu, and Christopher Amato. LLM collaboration with multi-agent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 32150–32158, 2026. doi: 10.1609/aaai.v40i38.40487.

Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip H. S. Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. MALT: Improving reasoning with multi-agent LLM training. In *Conference on Language Modeling*, 2025.

Chanwoo Park, Seungju Han, Xingzhi Guo, Asuman E. Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. MAPoRL: Multi-agent post-co-training for collaborative large language models with reinforcement learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 30215–30248, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1459.

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

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

Lloyd S. Shapley. A value for n-person games. In Harold W. Kuhn and Albert W. Tucker (eds.), *Contributions to the Theory of Games II*, pp. 307–317. Princeton University Press, Princeton, 1953.

Jianhong Wang, Yuan Zhang, Yunjie Gu, and Tae-Kyun Kim. SHAQ: Incorporating Shapley value theory into multi-agent Q-learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 5941–5954, 2022.

Tianwen Wei, Jian Luan, Wei Liu, Shuang Dong, and Bin Wang. CMATH: Can your language model pass Chinese elementary school math test? *arXiv preprint arXiv:2306.16636*, 2023.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *Conference on Language Modeling*, 2024.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025a. doi: 10.48550/arXiv.2412.15115.

Chih-Hsuan Yang, Tanwi Mallick, Le Chen, Krishnan Raghavan, Amal Gueroudji, Ian T. Foster, and Rajeev Thakur. Who gets the reward, who gets the blame? Evaluation-aligned training signals for multi-LLM agents. *arXiv preprint arXiv:2511.10687*, 2025b.

Yujie Zhao, Lanxiang Hu, Yang Wang, Minmin Hou, Hao Zhang, Ke Ding, and Jishen Zhao. Stronger-MAS: Multi-agent reinforcement learning for collaborative LLMs. In *International Conference on Learning Representations*, 2026.

A Extended Positioning

Table 7 compares the sources of credit information discussed in Section 2. It separates action-context comparisons, agent-removal comparisons, and learned coalition values. The entries summarize the principal credit mechanisms of each method. The controlled experimental setup used in this manuscript is described in Appendix E.1.

Method	Principal credit construction	Information used
Equal / MARFT-A (reference used here)	Equal allocation of the observed team reward	Team-level verifier score
C3 (Chen et al., 2026)	Contextual counterfactual advantage	Multiple context-matched action alternatives and a leave-one-out baseline
CCPO (Li et al., 2026b)	Agent-removal marginal credit	Realized and counterfactual outcomes with one agent removed
SHARP (Li et al., 2026a)	Shapley-based marginal-credit reward	Global accuracy, marginal-credit, and tool-process reward components, with trajectory-group advantage normalization
MASHAP-LLM / LLM-SHAP	Shapley allocation from predicted coalition values, rescaled to the observed team reward	Early verifier-labelled coalition rollouts, followed by frozen predictions from masked full-team trajectories

Table 7: Conceptual comparison of credit construction. Method descriptions follow the cited papers. The Equal / MARFT-A row denotes the reference allocation used in our experiments.

Relation to learned attribution. The design draws on both coalition-based credit assignment in reinforcement learning and amortized attribution in model explainability (Section 2). In our sequential protocol, the reference value of a coalition is defined by running its participating agents under an explicit absence convention. After fitting, the estimator instead predicts from masked views of the full-team trajectory. Masking does not reproduce a coalition execution, so these predictions and the resulting rewards require their own empirical evaluation. Shapley values are computed exactly from the predicted table for teams of up to three agents, while the larger-team variants use sampled permutations as described in Section 4.4.

B Detailed Notes for Reported Tables

Table 20. The table reports coalition-score variation and learned allocation nonuniformity. A task is *verifier-differentiating* if some non-empty, answer-capable strict subcoalition receives a verifier score different from the grand coalition's. The empty coalition and coalitions with no answer-capable agent are excluded from this restricted statistic. This score-variation event is distinct from unequal exact Shapley allocation. Appendix I.1 gives counterexamples. A task is LLM-SHAP-*differentiating* if the normalized learned reward vector differs from equal allocation by more than the tolerance $\epsilon_{\text{diff}} = 0.1$ of Appendix I.5. Appendix I.5 defines the diagnostic quantities and explains the distinction between score variation and allocation nonuniformity.

Table 1. Mathematical reasoning scores are Pass@1 accuracy. Fine-tuned methods report mean $\pm$ sample standard deviation over five seeds. For mathematical accuracy, each seed's percentage is rounded to two decimal places before averaging across seeds, and displayed means are then rounded to two decimal places. The upper block shows zero-shot SFT backbones, while the lower blocks compare post-training algorithms under MATH and CMATH training. Best results are bolded.

Table 2. Code generation results separate held-out Codeforces (ID) from MBPP-test and MBPP+ (OOD) for two and three agents. The OOD protocol evaluates the same final Codeforces-trained checkpoints, without additional training. All three columns use the Codeforces mean unit-test pass-rate definition. Trained methods use five-seed mean $\pm$ SD, and SFT is a direct-inference reference. Dataset sources and split sizes appear in Appendix E.3.1.

Table 3. Ablation on how the coalition-value table is obtained. The learned, verifier, and static variants share the Shapley allocation layer. L and CF-train denote learned value estimation and counterfactual-rollout supervision before freezing. *Role prior* substitutes the fixed per-role share of Table 28 for the predicted Shapley vector, so credit is non-uniform but identical on every task and no

coalition value is computed. The normalized fixed shares $\boldsymbol{w}$ sum to one and give allocated rewards $r_{b,i} = R_b w_i$. The Codeforces column uses its own pipeline and share vector. Equal allocation is repeated from Tables 1 and 2 as the reference floor.

Table 5. Four- and five-agent systems with sampled-coalition LLM-SHAP, compared with C3 under common task and evaluation conditions on MATH500, CMATH, GSM8K, and Codeforces. The four-agent system is Decomposer–Planner–Executor–Verifier, and the five-agent system adds a Reflector. Shapley credits are estimated from the learned coalition-value game by Monte Carlo permutation sampling with $K = 20$ orders and five seeds per cell. The permutation workload is $\mathcal{O}(Kn)$ marginal terms, compared with $\mathcal{O}(2^n)$ entries for enumeration. Appendix I.3 states the conditional error bound relative to exact allocation of the same learned game.

Table 6. (a) Frozen LLM-SHAP against coalition re-rollout reference values on trajectories logged *after* freezing, including those from the final trained policies. These were used neither for fitting the estimator nor for setting its freezing point. (b) Symmetric label flips applied only to the strict-subcoalition labels used to fit LLM-SHAP, with grand-coalition reward, evaluator, backbone, rollout budget and all other settings unchanged. Trained on MATH and evaluated on MATH500, $n = 3$, five seeds. The C3 reference is 48.12.

C Properties of Credit Allocation

This appendix characterizes the allocation used by MASHAP-LLM and its sensitivity to value-prediction errors. For a fixed task and policy snapshot, write v for the verifier-induced reference game, with $v(\emptyset) = 0$, $v(\mathcal{N}) = R_b$, and $0 \le v(C) \le R_{\max}$. Let $\hat{v}$ be the learned game. Denote the reference Shapley vector by $\boldsymbol{\phi}^* = \boldsymbol{\phi}(v)$ and the predicted raw credits by $\hat{\boldsymbol{\phi}} = \boldsymbol{\phi}(\hat{v})$. Take the normalization floor $\tau_{\text{floor}} > 0$. The analysis concerns a specified reference game or a fixed training batch.

C.1 Budget Preservation and Boundary Cases

Shapley efficiency gives $\sum_i \phi_i^* = R_b$ and $\sum_i \hat{\phi}_i = \hat{v}(\mathcal{N}) - \hat{v}(\emptyset) = \hat{\Sigma}_b$. On the rescaling branch of Eq. (4), summing $r_{b,i} = R_b \hat{\phi}_i / \hat{\Sigma}_b$ gives $\sum_i r_{b,i} = R_b$. On the fallback branch, each role receives R_b/n, so the same identity holds. Complete shared permutations preserve the learned endpoint difference before averaging, as detailed in Appendix I.3.1.

Budget preservation does not ensure nonnegative individual credits. For a failed task, $R_b = 0$ and the particular rescaling/fallback rule in Eq. (4) returns zero for every role. This differs from exact Shapley credit, which can contain positive and negative entries summing to zero. For example, $v(\emptyset) = v(\{1\}) = v(\{1,2\}) = 0$ and $v(\{2\}) = 1$ give $\boldsymbol{\phi}^* = (-1/2, 1/2)$. Thus the zero signal on failures follows from the implemented rule, rather than from budget preservation alone.

C.2 Value Error and Rescaling Sensitivity

Proposition 1 (Propagation of a uniform coalition-value error)**.** *Suppose $|\hat{v}(C) - v(C)| \le \varepsilon_{\text{est}}$ for every coalition C. Then exact Shapley evaluation of $\hat{v}$ satisfies*

$$|\hat{\phi}_i - \phi_i^*| \le 2\varepsilon_{\text{est}}, \qquad |\hat{\Sigma}_b - R_b| \le 2\varepsilon_{\text{est}}. \tag{7}$$

On a task satisfying the rescaling branch of Eq. (4)*, define $d_b = \max\{\tau_{\text{floor}}, R_b - 2\varepsilon_{\text{est}}\}$. The normalized credit obeys*

$$|r_{b,i} - \phi_i^*| \le \beta_b := 2\varepsilon_{\text{est}} + \frac{2\varepsilon_{\text{est}}(R_{\max} + 2\varepsilon_{\text{est}})}{d_b}. \tag{8}$$

On the fallback branch, the separate bound is $|R_b/n - \phi_i^*| \le R_b/n + R_{\max}$.

Proof. Each marginal-contribution error is at most $2\varepsilon_{\text{est}}$, and the nonnegative Shapley weights sum to one, giving the first bound. The sum error involves only the grand and empty coalition values, giving the second bound. The branch conditions imply $|\hat{\Sigma}_b| \ge d_b$, while bounded reference rewards

imply $|\phi_i^*| \le R_{\max}$ and hence $|\hat{\phi}_i| \le R_{\max} + 2\varepsilon_{\text{est}}$. Consequently,

$$|r_{b,i} - \phi_i^*| \le |\hat{\phi}_i - \phi_i^*| + \frac{|R_b - \hat{\Sigma}_b|}{|\hat{\Sigma}_b|}|\hat{\phi}_i|,$$

which proves Eq. (8). The fallback bound follows from the triangle inequality. □

The condition is a uniform error over all coalitions, not an empirical MAE or RMSE. The denominator also matters: rescaling can amplify small value errors when the predicted endpoint difference is small, and the fallback bound can be loose. The same reference game must be used for both predictions and errors. Changing from re-rollout contexts to masked contexts does not establish the assumed fidelity.

C.3 Sensitivity on a Fixed Training Batch

Fix a finite rollout batch, its empirical distribution μ_k, and policy snapshot $\boldsymbol{\theta}_k$. Assume each log policy is differentiable at the snapshot. With $z = (q, \boldsymbol{a})$, hold the contexts and credits fixed and define the role-specific surrogate

$$\mathcal{S}_i^{(k)}(\theta_i; \boldsymbol{r}) = \mathbb{E}_{z\sim\mu_k}[r_i(z) \log \pi_{\theta_i}(a_i \mid c_i)]\,. \tag{9}$$

Its derivative at the snapshot is $g_{k,i}[r] = \mathbb{E}_{\mu_k}[r_i s_{k,i}]$, where $s_{k,i} = \nabla_{\theta_i} \log \pi_{\theta_i}(a_i \mid c_i)|_{\theta_i=\theta_{k,i}}$. This derivative is defined for a finite greedy-generated batch without a stochastic on-policy sampling assumption.

For two allocations $\boldsymbol{r}, \boldsymbol{r}'$ on the same batch, assume the complete-response scores satisfy $\|s_{k,i}(z)\| \le G$. The stacked directions then satisfy

$$\|\boldsymbol{g}_k[r] - \boldsymbol{g}_k[r']\|_2 \le G\mathbb{E}_{\mu_k}\|\boldsymbol{r} - \boldsymbol{r}'\|_2. \tag{10}$$

Indeed, the stacked per-sample difference has norm at most $G\|\boldsymbol{r} - \boldsymbol{r}'\|_2$. Taking expectations proves the bound. It also applies to fixed advantages. With a common baseline their difference equals the credit difference. With different baselines, the baseline difference must be included.

At the rollout snapshot, the raw PPO actor derivative agrees with this score direction after replacing rewards by fixed advantages. This is a local sensitivity bound for a fixed batch at the policy snapshot.

D Algorithmic Details

Team-rollout helper in Algorithm 1. GreedyTeamRollout visits roles $i = 1, \dots, n$ in order, forms $c_{b,i} = (q_b, a_{b,1}, \dots, a_{b,i-1})$ with the role prompt, and generates $a_{b,i} = \text{Greedy}(\pi_{\theta_{i,\text{old}}}, c_{b,i})$. It records $\ell_{b,i} = \log \pi_{\theta_{i,\text{old}}}(a_{b,i} \mid c_{b,i})$ and $V_{b,i} = V_\omega(c_{b,i})$, returning the full trajectory and Ω_b. The pseudocode below shows full coalition enumeration for the small-team setting. The sampled extension evaluates the coalitions visited by the shared permutations and averages their marginal contributions, as specified in Section 4.4.

Prediction, fitting, and freezing schedule. Algorithm 1 uses a fixed behavior-policy snapshot throughout each policy update. Within block B, the current estimator first predicts the allocation table. If fitting is active, it is then updated on the new labeled examples and checked for freezing. The rewards already stored for the policy update remain fixed. After freezing, only context construction and value prediction remain active in this block. The four subroutines CoalTrain, CoalInfer, InferV, and FitV are given in Algorithms 2–5.

For $n \le 3$, we enumerate all 2^n coalitions. Early fitting requires coalition re-rollouts and verifier labels, while frozen allocation requires one full-team rollout, its verifier evaluation, and batched estimator predictions over the coalition masks. This removes strict-subcoalition verifier calls but retains the estimator's computational cost. The timing comparison in Appendix E.9 reports lower full-run time than direct verifier-induced allocation under the matched measured setting.

For larger teams, the reported configuration uses K shared sampled agent permutations and averages their marginal contributions, as detailed in Appendix I.3. The coalition contexts along those permutations are obtained by re-rollout during fitting and by masking after freezing. The uncompressed sampled expression has Kn marginal terms, whereas full enumeration uses 2^n distinct coalition

Algorithm 2 CoalTrain: coalition-label construction before estimator freezing.

Input: Question q_b, grand trajectory $\boldsymbol{a}_b^{\mathcal{N}}$, reward R_b, verifier Verifier, snapshot $\boldsymbol{\theta}_{\text{old}}$
Output: Labeled coalition dataset $\mathcal{D}_b^{\text{coal}}$

$\mathcal{D}_b^{\text{coal}} \leftarrow \emptyset$
for each coalition $C \subseteq \mathcal{N}$ **do**
  if $C = \mathcal{N}$ **then**
    $\boldsymbol{a}_b^C \leftarrow \boldsymbol{a}_b^{\mathcal{N}}; \quad y_C \leftarrow R_b$
  else if $C = \emptyset$ or C lacks the designated final-answer role **then**
    $\boldsymbol{a}_b^C \leftarrow \textsc{AbsTraj}(q_b, C)$
    $y_C \leftarrow 0$
  else
    $\boldsymbol{a}_b^C \leftarrow \textsc{CFRollout}(q_b, C; \boldsymbol{\theta}_{\text{old}})$
    $y_C \leftarrow \text{Verifier}(q_b, \boldsymbol{a}_b^C)$
  $x_C \leftarrow \textsc{BuildCtx}(q_b, \boldsymbol{a}_b^C, C)$
  $\mathcal{D}_b^{\text{coal}} \leftarrow \mathcal{D}_b^{\text{coal}} \cup \{(C, x_C, y_C)\}$
return $\mathcal{D}_b^{\text{coal}}$

Algorithm 3 CoalInfer: mask-conditioned context construction after estimator freezing.

Input: Question q_b, grand trajectory $\boldsymbol{a}_b^{\mathcal{N}}$
Output: Unlabeled coalition dataset $\mathcal{D}_b^{\text{coal}}$

$\mathcal{D}_b^{\text{coal}} \leftarrow \emptyset$
for each coalition $C \subseteq \mathcal{N}$ **do**
  Let $m_C \in \{0,1\}^n$ be the binary coalition mask induced by C
  $x_C \leftarrow \textsc{MaskCtx}(q_b, \boldsymbol{a}_b^{\mathcal{N}}, m_C)$
  $y_C \leftarrow \text{None}$
  $\mathcal{D}_b^{\text{coal}} \leftarrow \mathcal{D}_b^{\text{coal}} \cup \{(C, x_C, y_C)\}$
return $\mathcal{D}_b^{\text{coal}}$

entries. Repeated sampled coalitions can be reused, so these counts alone do not establish a computational saving. A fixed K need not be cheaper for four or five agents. The choice changes the approximation and cost of the Shapley stage, while preserving the component boundary in Algorithm 1.

D.1 LLM-SHAP Component Subroutines

This subsection summarizes the main subroutines used by LLM-SHAP, including coalition-label construction before estimator freezing, mask-conditioned coalition-context construction after freezing, allocation-time value inference, and estimator fitting.

D.2 Shapley Allocation and Reward Rescaling in MASHAP-LLM

This deterministic stage is downstream of the learned LLM-SHAP component. Algorithm 6 converts the predicted coalition-value table into raw Shapley credits and applies budget-preserving normalization with a numerical fallback.

D.3 Verifier-Induced Coalition Values for Supervision and Ablation

This subsection provides the implementation details for the verifier-induced supervision and ablation reference mentioned in Section 4.1. Verifier-induced coalition values are used to construct supervised labels for LLM-SHAP during early estimator fitting and to instantiate non-learned ablation variants. They are not the frozen allocation-time mechanism in the reported MASHAP-LLM results. After estimator freezing, allocation uses masked contextual coalition-value prediction instead of strict-subcoalition rollouts or verifier calls.

For early LLM-SHAP supervision and the verifier-induced ablation, we acquire v^{env} through environment-level coalition-conditioned rollouts. For each coalition C, agents in C participate in the rollout, while agents outside C are replaced by the standard absence message. An answer-capable role is the role designated to submit the final answer under the task protocol: Actor for two-agent mathematics, Coder for two-agent code, and Reviewer for both three-agent settings. Larger teams

Algorithm 4 INFERV: allocation-time LLM-SHAP value inference.

Input: Coalition dataset $\mathcal{D}_b^{\text{coal}}$, estimator $\hat{v}_\psi$
Output: Predicted coalition-value table $\widehat{\mathcal{V}}_b$
1: **for** each $(C, x_C, y_C) \in \mathcal{D}_b^{\text{coal}}$ **do**
2: $\quad \widehat{\mathcal{V}}_b(C) \leftarrow \hat{v}_\psi(x_C)$
3: **return** $\widehat{\mathcal{V}}_b$

Algorithm 5 FITV: estimator fitting before estimator freezing.

Input: Coalition dataset $\mathcal{D}_b^{\text{coal}}$, estimator $\hat{v}_\psi$
Output: Updated estimator parameters ψ
1: $\mathcal{L}_\psi \leftarrow 0$
2: **for** each $(C, x_C, y_C) \in \mathcal{D}_b^{\text{coal}}$ with $y_C \neq \text{None}$ **do**
3: $\quad \mathcal{L}_\psi \leftarrow \mathcal{L}_\psi + (\hat{v}_\psi(x_C) - y_C)^2$
4: Update ψ by minimizing $\mathcal{L}_\psi$
5: **return** ψ

retain their configuration-specific final-answer role. In three-agent mathematics, a coalition without the Reviewer therefore has zero environment reference value. This convention does not force the upstream roles' Shapley credits to zero, since those roles can change the outcomes of coalitions containing the Reviewer. The grand coalition reuses the cached verifier reward R_b, and coalitions without an answer-capable agent receive value 0 by the absence-message convention. The verifier-induced ablation applies EXACTSHAPLEY directly to v^{env} and uses the resulting Shapley vector without learned-value normalization, since Shapley efficiency gives $\sum_i \phi_i = v^{\text{env}}(\mathcal{N}) = R_b$.

Decoding of the coalition re-rollouts. CFROLLOUT uses the same greedy decoder ($T = 0$) as the original rollout and the same frozen policy snapshot $\boldsymbol{\theta}_{\text{old}}$ that produced $\boldsymbol{a}_b$. Consequently, for a given task the 2^n values of v^{env} form one determinate table rather than a sample from a distribution over tables. This is what fixes the reference against which the estimator diagnostics are read: MAE_v and RMSE_v compare the frozen estimator's prediction $\hat{v}_\psi(x_C)$ against this re-rollout value $v^{\text{env}}(C)$, and MAE_ϕ compares the credit vector derived from the learned coalition-value table against the credit vector derived from the same verifier-induced reference game. All three quantities therefore measure estimator error alone and carry no decoding variance.

Why no hard-zero on frozen inference. Although Algorithm 2 uses verifier-derived $y_C{=}0$ as the supervised target for no-answer-capable coalitions during early fitting, Algorithm 6 does not clamp $\hat{v}(C)$ to zero at frozen inference time. The learned estimator may produce non-zero contextual shaping estimates because it conditions on mask-conditioned contexts built from the grand-coalition trajectory. These estimates function as Shapley-shaped learning signals, and their reliability is assessed through the mechanism diagnostics of Appendix I.5.

D.4 CONDITIONAL VERIFIER-INDUCED SHAPLEY ABLATION AND STATIC MASKING

The static-masked Shapley variant replaces counterfactual rollout with a one-shot construction: keep a_i for $i \in C$ from the realized grand-coalition trajectory, replace a_i for $i \notin C$ with the standard absence message, and verify the composed answer. No re-rollouts are required. This variant still constructs a full static-masked coalition-value table and applies the Shapley allocator.

E IMPLEMENTATION DETAILS

This appendix describes the main experiments and core ablations. Within each task family, methods share the task split, backbone, role prompts, LoRA configuration, random seeds, final-checkpoint selection rule, and evaluation protocol. Baselines retain their method-specific training sampling and optimization. The greedy training settings and PPO analysis for MASHAP-LLM are identified separately below.

Algorithm 6 ALLOC: Shapley allocation with reward normalization.

Input: Predicted coalition-value table $\widehat{\mathcal{V}}_b$, realized reward R_b, floor τ_{floor}
Output: Budget-preserved agent-level reward vector $\boldsymbol{r}_b$

$\hat{\boldsymbol{\phi}} \leftarrow \textsc{ExactShapley}(\widehat{\mathcal{V}}_b, \mathcal{N})$
$\hat{\Sigma} \leftarrow \sum_{i=1}^{n} \hat{\phi}_i$
if $|\hat{\Sigma}| \geq \tau_{\text{floor}}$ and $\text{sign}(\hat{\Sigma}) = \text{sign}(R_b)$ **then**
    $\boldsymbol{r}_b \leftarrow R_b \cdot \hat{\boldsymbol{\phi}} / \hat{\Sigma}$
else
    $\boldsymbol{r}_b \leftarrow (R_b/n) \cdot \mathbf{1}$
return $\boldsymbol{r}_b$

Algorithm 7 Verifier-induced coalition-value acquisition for supervision and ablation.

Input: Question q_b, grand coalition trajectory $\boldsymbol{a}_b$, reward R_b, verifier Verifier, snapshot $\boldsymbol{\theta}_{\text{old}}$
Output: Verifier-induced coalition-value table v^{env}

for each coalition $C \subseteq \mathcal{N}$ **do**
    if $C = \mathcal{N}$ **then**
        $v^{\text{env}}(C) \leftarrow R_b$
    else if $C = \emptyset$ or C lacks the designated final-answer role **then**
        $v^{\text{env}}(C) \leftarrow 0$
    else
        $\boldsymbol{a}_b^C \leftarrow \text{CFROLLOUT}(q_b, C; \boldsymbol{\theta}_{\text{old}})$
        $v^{\text{env}}(C) \leftarrow \text{Verifier}(q_b, \boldsymbol{a}_b^C)$
return v^{env}

E.1 FAIR COMPARISON PROTOCOL

Here, a controlled comparison means common task, backbone, role-prompt, adapter, and evaluation conditions. C3, CCPO, and SHARP retain the sampling, credit construction, normalization, and policy-update mechanisms of their original methods. Their training samplers are distinct from the single-trajectory greedy sampler used by MASHAP-LLM. All methods use greedy decoding with one full-team trajectory per problem at evaluation.

The main performance comparisons use five random seeds, $3, 13, 23, 33, 43$, and report mean $\pm$ sample SD. Paired replication and component controls use separate seed sets, 24 for each mathematical training domain and 12 for Codeforces (Appendix K.1). These records are not pooled with the five-seed results.

Checkpoint selection. Each run in the main performance comparisons is evaluated at its fixed final update, and the five per-seed results are then averaged. The rule is identical for every compared method, and the single final checkpoint of a run is evaluated on all test benchmarks, so each trained-method result in Tables 1 and 2 averages five models rather than five maxima. The held-out buffer of Appendix E.8.1 holds coalition labels and determines when LLM-SHAP is frozen. It is an estimator-level quantity.

Baseline-specific hyperparameters belong to the corresponding algorithms. The normalization floor and safeguard in Eq. (4) apply to MASHAP-LLM and ablations explicitly using that allocation rule. They are not generic replacements for a baseline's original normalization. **Baseline-specific training mechanisms.** The following descriptions identify the algorithms in the cited baseline papers. Their auxiliary training samples and update rules are separate from the MASHAP-LLM settings in Tables 10–11.

C3. C3 (Chen et al., 2026) freezes a behavior-policy snapshot and collects decision contexts from reference trajectories. At each selected role/context pair, it samples multiple candidate messages and regenerates downstream collaboration under a fixed continuation policy and generation protocol. If candidate j has c_j replays with mean return $\bar{R}_j$, its credit is

$$A_j = \bar{R}_j - \frac{\sum_{k \neq j} c_k \bar{R}_k}{\sum_{k \neq j} c_k}.$$

The comparison uses at least two candidates within the same context. These context–message–credit tuples enter the original clipped PPO objective with reference-policy KL regularization. A fixed

continuation distribution fixes the policy and sampling protocol, not the downstream response text. Candidate generation is therefore distinct from greedy final evaluation.

CCPO. CCPO (Li et al., 2026b) constructs marginal credit $\Delta_i = R_{\text{all}} - R_{\text{without } i}$, normalizes contribution statistics using moving estimates, applies bounded reward shaping, and forms within-prompt advantages for clipped policy optimization. Its two-agent Think–Solve instantiation additionally samples the Solver without the Thinker's response. The Thinker uses marginal credit, while the Solver combines collaborative and independent reward signals through a history-dependent gate. These cooperative and counterfactual samples are not restricted by MASHAP-LLM's one-trajectory training convention.

SHARP. SHARP (Li et al., 2026a) combines task accuracy, marginal-credit, and tool-process components in an agent-specific reward. Worker credit compares the realized outcome with a masked-worker outcome, while planner credit aggregates positive worker contributions. Multiple trajectories are sampled per query. For agent m, the group-relative advantage is

$$\hat{A}_{j,m} = \frac{\bar{R}_{j,m} - \mu_m}{\sigma_m + \delta},$$

where μ_m and σ_m summarize the relevant sampled group. The original clipped importance-ratio objective aggregates over trajectories and participating agents. This normalization is distinct from the MASHAP-LLM reward-budget rescaling.

Equal allocation and allocation ablations. The equal-allocation reference gives each of the n roles R_b/n. The verifier-induced and static-masked variants use the coalition constructions in Appendices D.3–D.4. These controls use the MASHAP-LLM policy-training scaffold to examine allocation choices, while the other baselines retain their own training procedures.

Sampling and budget units. Training-task exposure, full-team trajectories, auxiliary replay branches, generated tokens, and verifier calls are distinct quantities. C3 can reuse a prefix when evaluating a candidate, whereas grouped methods generate several trajectories for one question. Equal task-batch sizes or update counts consequently do not establish equal total generation or verification cost. The main comparisons control task, model, and evaluation conditions. The timing comparison in Appendix E.9 concerns the explicitly named pipelines.

E.2 Effect-Size Context

Effect-size context. The math and Codeforces gains in Tables 1 and 2 compare complete training methods under common task, backbone, adapter, prompt, and evaluation conditions. Sampling, advantage construction, normalization, and policy updates retain their method-specific definitions.

In the main performance comparisons, relative to the strongest baseline in each setting (C3 or CCPO), MASHAP-LLM achieves average gains of 2.51 and 4.36 percentage points under the dual-agent and triple-agent settings, respectively. Each average equally weights nine evaluation settings: the six mathematical train–test settings in Table 1 and the held-out Codeforces, MBPP-test, and MBPP+ evaluations in Table 2. The four/five-agent averages cover only the four datasets in Table 5, using C3 as comparator. On mathematical reasoning, the absolute gains range from 1.71 to 7.16 points, while on Codeforces the gains are 1.05 and 1.64 points under the dual-agent and triple-agent settings, respectively. Within the mathematical and held-out Codeforces settings, the largest relative margins over the strongest non-MASHAP-LLM baseline reach 15.0% on CMATH→MATH500 and 34.0% on Codeforces.

E.3 Task Setup and Evaluation

Table 8 summarizes the task-level setup. Mathematical reasoning experiments use Qwen2.5-Coder-3B-Instruct as the backbone, while Codeforces experiments use Qwen2.5-3B-Instruct. Mathematical models are trained on MATH or CMATH and evaluated on MATH500, CMATH, and GSM8K. Codeforces models are trained on the Codeforces training split and evaluated on held-out Codeforces. Their two/three-agent checkpoints are additionally evaluated on MBPP-test and MBPP+ without further training (Appendix E.3.1).

Item	Mathematical Reasoning	Code Generation
Backbone	Qwen2.5-Coder-3B-Instruct	Qwen2.5-3B-Instruct
Training data	MATH / CMATH	Codeforces
Test data	MATH500, CMATH, GSM8K	Held-out Codeforces, plus MBPP-test and MBPP+ (OOD, $n = 2, 3$)
Dataset size	MATH: 7,500 / 500 CMATH: 600 / 1,098 GSM8K: test-only, 1,319	Codeforces: 6,610 / 393 MBPP-test: test-only, 500 MBPP+: test-only, 378
Data source	Public mathematical reasoning benchmarks	Public Codeforces problems, official MBPP and EvalPlus repositories
Split protocol	MATH uses the official train split for training and MATH500 (drawn from the official test split) for testing. CMATH provides no train split, so the 600 training problems are its validation split and the 1,098 test problems are its test split. The two are disjoint. GSM8K is used for testing only.	Codeforces: processed verifiable split, MBPP-test: official test split, MBPP+: official 378-task release
Evaluation generation	Greedy decoding, temperature $T = 0$, one sample per problem	Greedy decoding, temperature $T = 0$, one sample per problem
Evaluation size	All problems of each benchmark (MATH500: 500, CMATH: 1,098, GSM8K: 1,319)	All 393 Codeforces, 500 MBPP-test, and 378 MBPP+ problems
Evaluation metric	Pass@1 accuracy: number of correct problems divided by the number of problems	Average unit-test pass rate: for each problem the fraction of unit tests passed, averaged over problems. A program that fails to compile scores 0
Answer extraction / execution	Boxed answer from Actor ($n = 2$) or Reviewer ($n = 3$)	Compile and execute the final program from Coder ($n = 2$) or Reviewer ($n = 3$)
Correctness check	Exact match	Unit tests
Task-level validation split	None	None
Checkpoint selection	Final checkpoint, no test-set checkpoint selection	Final checkpoint, no test-set checkpoint selection

Table 8: Task-level setup and evaluation protocol. Mathematical reasoning uses public benchmarks and reports Pass@1 accuracy with boxed-answer extraction and exact-match evaluation. GSM8K, MBPP-test, and MBPP+ are used only for testing. All evaluation decoding is greedy at $T = 0$ with one sample per problem. Mathematical Pass@1 on N problems has the form k/N. Each run uses a fixed final-checkpoint selection rule and that checkpoint is evaluated on all test benchmarks. The held-out buffer of Appendix E.8.1 contains coalition labels for estimator freezing only. Codeforces uses a processed verifiable split from public programming problems and evaluates generated programs by compilation and unit tests.

E.3.1 OOD CODE EVALUATION

Data sources and scope. MBPP-test uses the 500 official test problems (task IDs 11–510) from the MBPP repository (Austin et al., 2021). MBPP+ uses the 378-task release from the EvalPlus repository (Liu et al., 2023), with expanded tests for MBPP-derived tasks. The two evaluations therefore share a task family. Both are test-only OOD evaluations relative to Codeforces training, and neither introduces an additional training set.

Checkpoints and evaluation. For $n = 2, 3$, each trained method uses its existing final Codeforces checkpoint, while SFT uses the corresponding unchanged backbone. No additional policy or estimator training is performed. Decoding and scoring follow the Codeforces configuration: greedy generation at $T = 0$, one team trajectory per problem, and the mean fraction of unit tests passed, averaged over problems and expressed as a percentage. The benchmark-specific test suites are used for execution, and compilation failures receive zero. The OOD protocol uses the same five training seeds and final-checkpoint selection rule. Four/five-agent evaluation remains limited to MATH500, CMATH, GSM8K, and Codeforces.

Aggregation scope. The $2.51/4.36$-percentage-point averages equally weight all nine evaluation settings: six mathematical train–test settings, held-out Codeforces, MBPP-test, and MBPP+. Figure 3

displays the corresponding setting-wise gains over the strongest controlled baseline in each setting. MBPP-test and MBPP+ each contribute one setting to the average, although they share a task family.

E.4 Backbone, LoRA, and Agent Profiles

All agents in the same multi-agent system use the same base model family but different role-specific LoRA adapters. This keeps the base model capacity fixed while allowing role specialization through adapter parameters.

Item	Value
LoRA rank r	8
LoRA α	16
LoRA dropout	0.05
LoRA target modules	`q_proj`, `v_proj`
Adapter design	Role-specific LoRA adapters with a shared base model
Gradient checkpoint steps	1
Orthogonal initialization	True

Table 9: LoRA and adapter configuration.

E.5 Rollout and Generation Settings

Table 10 reports MASHAP-LLM training-rollout and coalition-generation settings for mathematics and Codeforces. Baselines use the sampling procedures described in Appendix E.1. The common evaluation protocol is in Table 8.

Setting	Math	Codeforces
Context window	2,456	4,096
Max new tokens	512	1,024
Decoding	Greedy ($T = 0$)	Greedy ($T = 0$)
Samples per problem	1	1

Table 10: MASHAP-LLM rollout and generation settings. Each agent acts once in role order and the verifier scores the completed transcript (Section 3). Greedy decoding is used for MASHAP-LLM training trajectories and the coalition re-rollouts of Appendix D.3. Baseline-specific training sampling is separate.

E.6 Optimization Hyperparameters

Table 11 reports the PPO configuration for MASHAP-LLM and the allocation ablations using its policy-training scaffold, including equal allocation and verifier-induced Shapley. Baseline-specific objectives and normalization follow Appendix E.1. This table does not impose a common PPO/critic configuration on C3, CCPO, and SHARP.

E.7 Clipping Regime of the Two Task Settings

For the MASHAP-LLM mathematical configuration in Table 11, a single full-batch actor gradient is evaluated before its parameter update. Provided rollout and gradient evaluation use identical likelihood conventions and model state, $\theta_i = \theta_{i,\mathrm{old}}$ at that evaluation and hence $\rho_{b,i} = 1$. Clipping is inactive at that point.

The MASHAP-LLM Codeforces batch is split into 20 minibatches. Later gradients may be evaluated after the policy has moved, so ratios can depart from one. Minibatch count alone does not establish that clipping occurs.

The fixed-batch interpretation in Appendix C.3 applies to the greedy-generated data at the rollout snapshot, while later minibatches use their actual likelihood ratios.

Hyperparameter	Math	Codeforces
Batch size	256	256
Policy updates	MATH: 8,000, CMATH: 20,000	5,000
Learning-rate warmup	None	None
Policy learning rate	1.0×10^{-6}	1.0×10^{-6}
Critic learning rate	1.0×10^{-5}	1.0×10^{-5}
PPO epochs	1	1
Mini-batches	1	20
PPO clip $\varepsilon_{\text{clip}}$	0.2	0.2
Entropy coefficient	0.001	0.001
Value loss coefficient	1.0	1.0
Max gradient norm	0.5	0.5
Clipped value loss	True	True

Table 11: Core PPO hyperparameters for MASHAP-LLM and its allocation ablations. Update counts and task-batch sizes in this table describe that scaffold. Method-specific auxiliary sampling is not counted as a single full-team rollout. The two mathematical training sets differ in size (7,500 MATH problems against 600 CMATH problems), so at these update counts each MATH problem appears in about 273 task batches on average and each CMATH problem in about 8,533 (about 31× as many). The CMATH column should therefore be read as a higher-training-intensity regime rather than as evidence of data efficiency. With one full-batch actor gradient evaluation at the rollout snapshot, the likelihood ratio is one at that evaluation, but it need not remain one after the parameter step. In Codeforces, later minibatches can activate clipping (Appendix E.7). LLM-SHAP estimator-specific settings are reported separately in Table 12.

E.8 LLM-SHAP Estimator and Reward Allocation

This subsection reports the estimator architecture, reward-allocation settings, estimator-freezing rule, and frozen allocation-time semantics of LLM-SHAP.

For $n \leq 3$, all 2^n coalitions are enumerated exactly during supervised fitting and all 2^n coalition masks are enumerated exactly during frozen inference. For $n \geq 4$ the enumeration is replaced by Monte Carlo permutation sampling with $K = 20$ sampled agent orders (Section 4.4). Appendix I.3 measures the resulting deviation from exact enumeration.

Item	Value
Coalition enumeration	Exact for $n \leq 3$ (all 2^n coalitions when fitting, all 2^n masks at frozen inference), Monte Carlo permutation sampling with $K = 20$ orders for $n \geq 4$
LLM-SHAP value head	1-layer linear head by default, $1024 \to 512 \to 1$ MLP in selected runs
LLM-SHAP estimator LR	1.0×10^{-5}
Budget-preserving normalization	Floored, sign-aware rescaling
Non-degenerate threshold τ_{floor}	$0.05R_{\max}$
Fallback rule	Equal allocation when rescaling is degenerate or sign-inconsistent
Early-stopping patience p_{shap}	3 validation windows by default
Minimum relative improvement δ_{shap}	10^{-3} by default
Validation-loss stabilizer ϵ_{loss}	10^{-8}
Validation check interval E_{val}	one estimator-fitting window by default

Table 12: LLM-SHAP estimator and reward-allocation settings.

For mathematical reasoning, the LLM-SHAP estimator uses Qwen2.5-Coder-3B-Instruct. For Codeforces, it uses Qwen2.5-3B-Instruct, matching the policy backbone used in the corresponding experiments. The backbone is paired with a trainable value head, implemented as either a one-layer linear head or a $1024 \to 512 \to 1$ MLP.

E.8.1 Estimator-freezing rule and frozen allocation semantics

The estimator-freezing rule λ_{stop} in Algorithm 1 determines when online fitting of the LLM-SHAP estimator is stopped. In this work, λ_{stop} is instantiated as validation-based early stopping on held-out verifier-derived coalition labels.

During early estimator fitting, each labeled coalition example is written as

$$(x_C, y_C), \qquad y_C = v^{\text{env}}_{q,\boldsymbol{a}}(C) = \text{Verifier}(q, \boldsymbol{a}^C),$$

where x_C is the coalition-conditioned context and y_C is the verifier-induced coalition value. Let $\mathcal{D}^{\text{val}}_{\text{coal}}$ be a held-out coalition-label buffer disjoint from the minibatches used by FITV. This estimator-level validation buffer is used only for freezing LLM-SHAP and is separate from task-level evaluation benchmarks. We evaluate the held-out estimator loss every E_{val} estimator-fitting windows. After the m-th validation check, we compute

$$\mathcal{L}^{(m)}_{\text{val}} = \frac{1}{|\mathcal{D}^{\text{val}}_{\text{coal}}|} \sum_{(x_C, y_C) \in \mathcal{D}^{\text{val}}_{\text{coal}}} \left(\hat{v}_{\psi_m}(x_C) - y_C\right)^2. \tag{11}$$

For the first validation evaluation, we set

$$\mathcal{L}_{\text{best}} \leftarrow \mathcal{L}^{(1)}_{\text{val}}, \qquad \psi^\star \leftarrow \psi_1.$$

For later validation evaluations, the relative improvement is

$$\text{Imp}^{(m)} = \frac{\mathcal{L}_{\text{best}} - \mathcal{L}^{(m)}_{\text{val}}}{\max(\mathcal{L}_{\text{best}}, \epsilon_{\text{loss}})}. \tag{12}$$

If $\text{Imp}^{(m)} > \delta_{\text{shap}}$, we update $\mathcal{L}_{\text{best}} \leftarrow \mathcal{L}^{(m)}_{\text{val}}$, store $\psi^\star \leftarrow \psi_m$, and reset the no-improvement counter. If $\text{Imp}^{(m)} \le \delta_{\text{shap}}$ for p_{shap} consecutive evaluation windows, we freeze the estimator, restore $\psi^\star$, and stop subsequent FITV updates.

After freezing, LLM-SHAP is used only for mask-conditioned allocation-time coalition-value prediction. Environment-level coalition-specific rollouts and strict-subcoalition verifier calls are stopped after freezing. Specifically, COALINFER enumerates coalition masks and constructs $x^{\text{infer}}_{b,C}$ for estimator inference, but it does not generate $\boldsymbol{a}^C_b$ for strict subcoalitions and does not request strict-subcoalition verifier labels. The verifier is called once on the realized grand-coalition trajectory to obtain R_b for budget preservation. Importantly, R_b is not inserted as the coalition value $\hat{v}_\psi(\mathcal{N})$. The grand-coalition entry is also predicted by the frozen estimator. Consequently, each post-freeze training example with $n \le 3$ uses one environment-level grand-coalition rollout and one grand-coalition verifier reward, while still enumerating 2^n masks for batched LLM-SHAP forward predictions rather than 2^n environment rollouts or verifier evaluations.

The estimator diagnostics report MAE_v, RMSE_v, and raw-credit MAE_ϕ on logged contexts. They do not by themselves establish the error of the final rescaled training rewards. Paired controls separately evaluate input and freezing noninferiority.

Compute. All experiments are run on NVIDIA H200 GPUs. The wall-clock training time varies substantially across tasks, agent counts, and ablation variants. Each reported configuration typically requires approximately 100–600 wall-clock hours on our multi-GPU setup. In multi-agent runs, agents are typically deployed across separate GPUs, so these numbers report elapsed training time rather than accumulated GPU-hours.

E.9 Performance–Cost Trade-off

Early estimator fitting incurs coalition re-rollouts and value-head optimization. After freezing, allocation retains estimator inference but no longer performs strict-subcoalition environment generation or verification. The following timing comparison uses the same task, hardware, and training budget, with single-agent PPO as the normalization reference.

MASHAP-LLM takes approximately 5.8% less full-run time than verifier-induced allocation in this measured comparison, while costing more than single-agent PPO and static masking. The frozen-phase ratio is reported separately as $1.344\times$. The difference between this phase-level ratio and the full-run ratio is not an estimate of fitting overhead.

Training configuration	Scope	Relative time
Single-agent PPO	Baseline	1.000×
Static-masked	Full run	1.090×
MASHAP-LLM	Full run	1.473×
Verifier-induced	Full run	1.563×
MASHAP-LLM	Frozen phase	1.344×

Table 13: Relative timing under matched task, hardware, and training budget, with single-agent PPO as the baseline. The frozen-phase measurement is listed separately from full-run costs. Phase and full-run ratios are not subtracted to estimate fitting overhead. This normalization differs from an Equal / MARFT-A multi-agent reference.

F AGENT PROMPTS

This appendix reports the role prompts used by the two-agent and three-agent multi-agent systems. All compared methods use the same role prompts under the same task and agent-count setting. Their sampling and learning procedures retain the method-specific definitions in Appendix E.1. We report the system-level role prompts and the corresponding user prompt templates.

F.1 TWO-AGENT SETTING

F.1.1 MATHEMATICAL REASONING: REASONER → ACTOR

Role	Order	Answer-producing agent
Reasoner	1	No
Actor	2	Yes

Table 14: Two-agent mathematical reasoning pipeline.

Math 2-Agent Reasoner: system prompt

```
<|im_start|>system: 2 LLM agents (Reasoner -> Actor) collaborate
    step-by-step to solve math problems. Balanced pipeline: precise,
     concise, and cooperative. Prefer clear assumptions, stepwise
    work, and minimal but sufficient detail. Keep reasoning
    structured and avoid unsupported shortcuts.
You are the **Reasoner**. Analyze the original math problem before
    any computation. Identify the goal, known conditions, unknowns,
    constraints, and likely solution path. If previous actions or
    feedback are provided, summarize only the useful state. Provide
    the next best operation for the Actor, but do not complete the
    full solution or place a final answer in boxed form.
General rules: stay within your role, cite assumptions explicitly,
    do not fabricate conditions, and make your output useful to the
    next agent.<|im_end|>
```

Math 2-Agent Reasoner: user prompt template

```
Question:
{question}

Previous agent messages:
{history}

Please analyze the mathematical structure of the problem, list the
    useful conditions and constraints, and propose the next best
    operation for the Actor. Do not provide the final boxed answer.
```

Math 2-Agent Actor: system prompt

```
<|im_start|>system: 2 LLM agents (Reasoner -> Actor) collaborate
    step-by-step to solve math problems. Balanced pipeline: precise,
     concise, and cooperative. Prefer clear assumptions, stepwise
    work, and minimal but sufficient detail. Keep reasoning
    structured and avoid unsupported shortcuts.
You are the **Actor**. Carry out the Reasoner's plan and perform
    the main mathematical operations. Show necessary derivations,
    handle cases explicitly, and keep notation consistent. If you
    are the designated answer-producing agent in this configuration,
     provide the final result in \boxed{{}}; otherwise provide a
    provisional result for later agents to inspect.
General rules: stay within your role, cite assumptions explicitly,
    do not fabricate conditions, and make your output useful to the
    next agent.<|im_end|>
```

Math 2-Agent Actor: user prompt template

```
Question:
{question}

Previous agent messages:
{history}

Please execute the Reasoner's plan, show the necessary derivation
    steps, and provide the final answer in \boxed{} format when you
    are the answer-producing agent.
```

F.1.2 CODEFORCES: REASONER → CODER

Role	Order	Answer-producing agent
Reasoner	1	No
Coder	2	Yes

Table 15: Two-agent Codeforces pipeline.

Codeforces 2-Agent Reasoner: system prompt

```
<|im_start|>system: 2 LLM agents (Reasoner -> Coder) collaborate to
     solve Codeforces Python coding problems. Balanced pipeline:
    precise, concise, and cooperative. Prefer clear assumptions,
    stepwise work, and minimal but sufficient detail. Prioritize
    correctness, clarity, and asymptotic efficiency.
You are the **Reasoner**. Analyze the programming problem before
    implementation. Extract input format, output format, constraints
    , objective, and edge cases. Propose an efficient algorithm and
    complexity target. You may write pseudocode, but do not write
    actual Python code.
General rules: stay within your role, respect the problem
    constraints, avoid unsupported claims, and make your output easy
     for the next agent to use.<|im_end|>
```

Codeforces 2-Agent Reasoner: user prompt template

```
Problem statement:
{question}

Previous agent messages:
{history}

Please analyze the problem, extract constraints and edge cases, and
    propose an efficient algorithmic solution. You may include
    pseudocode, but do not write final Python code.
```

Codeforces 2-Agent Coder: system prompt

```
<|im_start|>system: 2 LLM agents (Reasoner -> Coder) collaborate to
    solve Codeforces Python coding problems. Balanced pipeline:
    precise, concise, and cooperative. Prefer clear assumptions,
    stepwise work, and minimal but sufficient detail. Prioritize
    correctness, clarity, and asymptotic efficiency.
You are the **Coder**. Implement the Reasoner's algorithm in Python.
    Produce complete runnable code with robust input parsing, clear
    control flow, and correct handling of edge cases. Avoid
    unnecessary abstractions. If you are the answer-producing agent
    in this configuration, place the full solution below 'Answer:'.
General rules: stay within your role, respect the problem
    constraints, avoid unsupported claims, and make your output easy
    for the next agent to use.<|im_end|>
```

Codeforces 2-Agent Coder: user prompt template

```
Problem statement:
{question}

Previous agent messages:
{history}

Please implement the final Python solution according to the
    Reasoner's plan. Put the complete program below the line
    starting with 'Answer:'.
```

F.2 Three-Agent Setting

F.2.1 Mathematical Reasoning: Reasoner → Actor → Reviewer

The three agents act once each in this order. The Reasoner plans the solution, the Actor derives candidate results, and the Reviewer checks and corrects the derivation within its own response before submitting the final boxed answer. The external verifier extracts and scores that answer. The third agent is called Verifier in the execution templates and Reviewer below, distinguishing it from the external scoring function.

Role	Order	Answer-producing agent
Reasoner	1	No
Actor	2	No
Reviewer	3	Yes

Table 16: Three-agent mathematical reasoning pipeline.

Math 3-Agent Reasoner: system prompt

```
<|im_start|>system: 3 LLM agents (Reasoner -> Actor -> Reviewer)
    collaborate to solve math problems. Balanced pipeline: precise,
    concise, and cooperative. Prefer clear assumptions, structured
    work, and minimal but sufficient detail. The Reviewer is the
    sole final-answer-producing agent.
You are the **Reasoner**. Analyze the original math problem before
    the main computation. Identify the goal, known conditions,
    unknowns, constraints, and likely solution path. If previous
    actions or feedback are provided, summarize only the useful
    state. Provide a clear plan for the Actor, including relevant
    case distinctions and conditions that must be checked. Do not
    complete the full solution, present a final answer, or use boxed
    -answer formatting.
General rules: stay within your role, state assumptions explicitly,
     do not fabricate conditions, and make your output useful to the
     next agent.<|im_end|>
```

Math 3-Agent Reasoner: user prompt template

```
Question:
{question}

Previous agent messages:
{history}

Please analyze the mathematical structure of the problem, identify
    the useful conditions and constraints, and propose a solution
    plan for the Actor. Highlight any necessary case distinctions or
     domain restrictions. Do not provide the full solution or a
    final boxed answer.
```

Math 3-Agent Actor: system prompt

```
<|im_start|>system: 3 LLM agents (Reasoner -> Actor -> Reviewer)
    collaborate to solve math problems. Balanced pipeline: precise,
    concise, and cooperative. Prefer clear assumptions, structured
    work, and minimal but sufficient detail. The Reviewer is the
    sole final-answer-producing agent.
You are the **Actor**. Carry out the Reasoner's plan and perform
    the main mathematical operations. Show necessary derivations,
    handle cases explicitly, and keep notation consistent. If the
    plan contains an error or omission, explain the necessary
    adjustment and continue when possible. Provide provisional
    candidate results for the Reviewer to inspect, clearly
    distinguishing established steps from unresolved assumptions or
    checks.
Label any proposed answer as "Candidate result:". You must not
    present it as the final answer, use a "Final Answer:" section,
    or use boxed-answer formatting. Even if the solution appears
    complete, leave final validation and final-answer presentation
    to the Reviewer.
General rules: stay within your role, state assumptions explicitly,
     do not fabricate conditions, and make your output useful to the
     next agent.<|im_end|>
```

Math 3-Agent Actor: user prompt template

```
Question:
{question}

Previous agent messages:
{history}

Please execute the Reasoner's plan and show the necessary
    derivation steps. Provide provisional candidate results and
    identify any unresolved conditions or checks for the Reviewer.
    Do not present a final answer or use boxed-answer formatting.
```

Math 3-Agent Reviewer: system prompt

```
<|im_start|>system: 3 LLM agents (Reasoner -> Actor -> Reviewer)
    collaborate to solve math problems. Balanced pipeline: precise,
    concise, and cooperative. Prefer clear assumptions, structured
    work, and minimal but sufficient detail. The Reviewer is the
    sole final-answer-producing agent.
You are the **Reviewer** and the final-answer-producing agent.
    Review the Actor's derivation and candidate results against the
    original problem, using the Reasoner's plan as context rather
    than as proof. Check algebra, logic, domains, theorem
    assumptions, case completeness, and whether the proposed results
     satisfy the problem statement. Check for extraneous or missing
    solutions where relevant. Use substitution, independent
    reasoning, or sanity checks when useful.
If an error or gap is found, identify the failed step, correct it,
    and complete the necessary reasoning before finalizing. Do not
    stop at requesting revision when you can resolve the issue
    yourself. Do not accept a candidate solely because earlier
    agents agree.
Start with "Verification:" and briefly state the decisive checks
    and any corrections. Then write "Final Answer:" and present the
    verified final result in \boxed{} format. Only you may output
    the final boxed answer. If a verified answer cannot be
    established, explicitly state what remains unresolved rather
    than inventing an answer.
General rules: stay within your role, state assumptions explicitly,
     do not fabricate conditions, and ensure the final answer
    addresses the original question.<|im_end|>
```

Math 3-Agent Reviewer: user prompt template

```
Question:
{question}

Previous agent messages:
{history}

Please verify the Actor's derivation and candidate results against
    the original problem. Check algebra, logic, domains, theorem
    assumptions, case completeness, and answer validity. Correct any
     errors and complete any missing reasoning that you can resolve.
     Start with "Verification:", then provide the verified final
    answer below "Final Answer:" in \boxed{} format.
```

F.2.2 CODEFORCES: REASONER → CODER → REVIEWER

Role	Order	Answer-producing agent
Reasoner	1	No
Coder	2	No
Reviewer	3	Yes

Table 17: Three-agent Codeforces pipeline.

Codeforces 3-Agent Reasoner: system prompt

```
<|im_start|>system: 3 LLM agents (Reasoner -> Coder -> Reviewer)
    collaborate to solve Codeforces Python coding problems. Balanced
     pipeline: precise, concise, and cooperative. Prefer clear
    assumptions, stepwise work, and minimal but sufficient detail.
    Prioritize correctness, clarity, and asymptotic efficiency.
You are the **Reasoner**. Analyze the programming problem before
    implementation. Extract input format, output format, constraints
    , objective, and edge cases. Propose an efficient algorithm and
    complexity target. You may write pseudocode, but do not write
    actual Python code.
General rules: stay within your role, respect the problem
    constraints, avoid unsupported claims, and make your output easy
     for the next agent to use.<|im_end|>
```

Codeforces 3-Agent Reasoner: user prompt template

```
Problem statement:
{question}

Previous agent messages:
{history}

Please analyze the problem, extract constraints and edge cases, and
     propose an efficient algorithmic solution. You may include
    pseudocode, but do not write final Python code.
```

Codeforces 3-Agent Coder: system prompt

```
<|im_start|>system: 3 LLM agents (Reasoner -> Coder -> Reviewer)
    collaborate to solve Codeforces Python coding problems. Balanced
     pipeline: precise, concise, and cooperative. Prefer clear
    assumptions, stepwise work, and minimal but sufficient detail.
    Prioritize correctness, clarity, and asymptotic efficiency.
You are the **Coder**. Implement the Reasoner's algorithm in Python.
     Produce complete runnable code with robust input parsing, clear
     control flow, and correct handling of edge cases. Avoid
    unnecessary abstractions. If you are the answer-producing agent
    in this configuration, place the full solution below 'Answer:'.
General rules: stay within your role, respect the problem
    constraints, avoid unsupported claims, and make your output easy
     for the next agent to use.<|im_end|>
```

Codeforces 3-Agent Coder: user prompt template

```
Problem statement:
{question}

Previous agent messages:
{history}

Please implement the Python solution according to the Reasoner's
    plan. Provide complete runnable code for the Reviewer to inspect
    .
```

Codeforces 3-Agent Reviewer: system prompt

```
<|im_start|>system: 3 LLM agents (Reasoner -> Coder -> Reviewer)
    collaborate to solve Codeforces Python coding problems. Balanced
     pipeline: precise, concise, and cooperative. Prefer clear
    assumptions, stepwise work, and minimal but sufficient detail.
    Prioritize correctness, clarity, and asymptotic efficiency.
You are the **Reviewer**. Review the implementation and all prior
    feedback. Check correctness, complexity, input/output behavior,
    edge cases, and Python-specific pitfalls. If changes are needed,
     provide the corrected full code. Start with 'Review:' and put
    the final code below 'Final Answer:'.
General rules: stay within your role, respect the problem
    constraints, avoid unsupported claims, and make your output easy
     for the next agent to use.<|im_end|>
```

Codeforces 3-Agent Reviewer: user prompt template

```
Problem statement:
{question}

Previous agent messages:
{history}

Please review the proposed algorithm and code. Check correctness,
    complexity, input/output behavior, and edge cases. If changes
    are needed, provide the corrected full Python code below 'Final
    Answer:'.
```

F.3 Coalition Absence Message

For coalition-value acquisition, agents outside the coalition are replaced by a fixed absence message. The same absence message is used across verifier-induced Shapley allocation, the MASHAP-LLM algorithm, and the corresponding ablations.

Absence message

```
[System: The {role} did not participate in this round.]
```

F.4 ABSENCE-MESSAGE NLL DIAGNOSTIC

Counterfactual coalition construction requires representing agents outside the coalition. Our implementation uses a fixed absence message for non-coalition agents. This deterministic intervention may introduce textual distribution shift if downstream agents react to the absence marker itself.

The diagnostic measures the likelihood of the final-answer agent's recorded response: Actor in two-agent mathematics, Coder in two-agent code, and Reviewer in both three-agent settings. For the same recorded response a_{out}, we compute token-normalized negative log-likelihood under its reference context c_{ref} and the corresponding absence-message context c_{abs}:

$$\begin{aligned} \Delta\text{NLL}_{\text{out}} &= \text{NLL}_{\text{abs}} - \text{NLL}_{\text{ref}}, \\ \text{NLL}_{\text{abs}} &= \text{NLL}(a_{\text{out}} \mid c_{\text{abs}}), \\ \text{NLL}_{\text{ref}} &= \text{NLL}(a_{\text{out}} \mid c_{\text{ref}}). \end{aligned} \tag{13}$$

A positive paired difference indicates higher NLL under the absence-message context. Table 18 reports the missing upstream roles using their task-specific names.

Setting	Missing role(s)	Pairs	Mean $\Delta\text{NLL}_{\text{out}}$	Seed-bootstrap 95% CI
MATH, $n = 2$	Reasoner	300	0.009371	$[0.003556, 0.015197]$
MATH, $n = 3$	Reasoner	100	0.003532	$[-0.003584, 0.010647]$
MATH, $n = 3$	Actor	100	0.012631	$[0.004226, 0.019672]$
MATH, $n = 3$	Reasoner + Actor	100	0.024509	$[0.017828, 0.031068]$
CMATH, $n = 2$	Reasoner	300	0.008979	$[0.006508, 0.011475]$
CMATH, $n = 3$	Reasoner	100	0.000533	$[-0.003014, 0.003678]$
CMATH, $n = 3$	Actor	100	0.007828	$[0.005136, 0.012508]$
CMATH, $n = 3$	Reasoner + Actor	100	0.017286	$[0.010839, 0.025183]$
Code, $n = 2$	Reasoner	300	0.008459	$[0.004223, 0.013300]$
Code, $n = 3$	Reasoner	100	0.000510	$[-0.000991, 0.002455]$
Code, $n = 3$	Coder	100	0.012826	$[0.008479, 0.016519]$
Code, $n = 3$	Reasoner + Coder	100	0.013290	$[0.010233, 0.016084]$

Table 18: Paired likelihood changes for the designated final-answer role under upstream absence interventions. Each pair evaluates the same recorded output under the reference and absence-message contexts.

Mean paired differences range from 0.000510 to 0.024509. For three-agent teams, all three Reasoner-only intervals include zero, while removing the Actor or Coder, or both upstream roles, gives positive intervals. These measurements describe response-likelihood changes. They do not isolate the effect of the absence marker from that of removing upstream information.

G ADDITIONAL ANALYSIS

MASHAP-LLM as the proposed allocation algorithm. The empirical math and Codeforces results reported in Tables 1 and 2 correspond to the early-fitted and then frozen MASHAP-LLM allocation algorithm. Verifier-induced coalition evaluation is used to construct supervised coalition labels and to instantiate controlled ablations, but it is not the frozen allocation algorithm. The motivation for LLM-SHAP is that sparse verifier feedback can make many coalitions indistinguishable under the verifier. By predicting coalition values from mask-conditioned contexts before applying exact Shapley allocation, frozen LLM-SHAP defines a learned contextual coalition-value game while preserving the realized system-level reward budget.

Zero coalition values and learned shaping. A coalition without an answer-capable role can have $v(C) = 0$, while its members retain nonzero Shapley credit through marginal contributions in larger coalitions. A zero standalone value therefore does not establish that the verifier underestimates a role's contribution. LLM-SHAP changes the learned coalition game by assigning non-zero contextual shaping estimates to such coalitions based on mask-conditioned contexts built from the grand-coalition trajectory, even when the verifier-derived training target is zero. We interpret these estimates as shaping signals rather than alternative verifier scores. This is the design motivation for why the inference rule of Algorithm 4 avoids hard-coding such coalitions to zero. This effect motivates the diagnostics in Appendix I, which evaluate whether the learned coalition-value estimator accurately approximates verifier-induced coalition values from frozen masked contextual predictions.

Method	MATH →MATH500	MATH →CMATH	MATH →GSM8K
Single-Agent SFT	35.40	79.23	66.57
Single-Agent PPO	36.44	80.24	68.45

Table 19: Single-agent RL control with MATH training, Pass@1 (%). SFT denotes direct inference without reinforcement fine-tuning, while PPO reports mean performance after reinforcement fine-tuning. Both rows use the single-agent setup and evaluation protocol described here.

Codeforces continuous pass-rate behavior. Codeforces naturally creates more non-zero partial coalition values than binary math tasks because each problem is evaluated using a unit-test suite. In binary math benchmarks, exact Shapley differs from Equal allocation only when a strict subcoalition succeeds. By contrast, in Codeforces, even when the fast packed test fails, per-test execution can yield a non-trivial coalition value $v(C) \in [0, 1]$. Table 20 reports restricted coalition-score variation, a different event from unequal exact Shapley allocation. Continuous scores alone do not establish an increase in the latter event.

In Table 2, MASHAP-LLM obtains positive margins on Codeforces in both agent configurations. In the dual-agent setting, MASHAP-LLM improves from 4.52% under MARFT-A / Equal to 5.79%, and from 4.33% under the SFT backbone to 5.79%. In the triple-agent setting, MASHAP-LLM improves from 4.46% under MARFT-A / Equal to 6.47%, and from 1.83% under the SFT backbone to 6.47% (Table 2).

Static-masked Shapley, SHARP, and agent-count considerations. The static-masked Shapley variant is included in the ablation study (Table 3). It reuses the realized grand-coalition trajectory and masks non-coalition agents without coalition-specific re-rollout, thereby testing whether post-hoc masking can replace counterfactual coalition rollout for full coalition-table Shapley allocation. This ablation is closely related to the masking semantics used by SHARP, but it differs in the credit operator: SHARP uses only the one-agent grand-coalition ablation score, whereas static-masked Shapley constructs a full static-masked coalition-value table for all coalitions and then applies the Shapley allocator. Static-masked Shapley has lower reported scores than verifier-induced Shapley in Table 3. **Sparse binary feedback and rollout-construction ablation.** Under binary verifiers $R \in \{0, 1\}$, if the verifier-induced coalition game reduces to a unanimity game in which the grand coalition succeeds and all strict subcoalitions fail, exact Shapley reduces to $\phi_i = 1/n$ for all indispensable agents. In this special case, Equal allocation coincides with the mathematically correct Shapley allocation implied by the sparse verifier. Thus, verifier-induced Shapley allocation is not guaranteed to provide denser role-wise credit under sparse final-answer feedback. It can differ from Equal only when the verifier-induced coalition table induces unequal average marginal contributions across agents. We use counterfactual rollout as the default verifier-derived coalition-value construction for early estimator supervision and verifier-induced ablation, but not for frozen LLM-SHAP allocation. The static-masking ablation, which reuses the realized grand-coalition trajectory and masks non-coalition agents without re-rolling, is reported in Table 3, with implementation details in Appendix D.4.

H Single-Agent RL Control

We report a single-agent PPO control with MATH training. This architecture control removes the role-specialized multi-agent decomposition while keeping the same backbone family, verifier, training data, LoRA configuration, optimizer setting, decoding protocol, and evaluation metrics.

Table 19 reports the single-agent scores. The reported PPO MATH500 mean is 36.44%. The two- and three-agent MASHAP-LLM systems score 52.28% and 54.44%, respectively (Table 1). The main credit-assignment comparisons use a common multi-agent topology, with each method retaining its own sampling, credit construction, and policy-update procedures.

I LLM-SHAP Mechanism Diagnostics

This appendix evaluates whether the early-fitted and frozen LLM-SHAP protocol provides reliable coalition-value estimates and useful allocation-time reward signals when frozen inference uses masked grand-trajectory contexts. The diagnostics address four questions: (i) whether the frozen estimator remains faithful to verifier-induced coalition values, (ii) whether prediction errors substantially

distort the downstream Shapley allocation, (iii) how often the denominator magnitude falls below the normalization floor, and (iv) how learned allocation nonuniformity relates to restricted verifier-score variation under sparse feedback. These are distinct events. Their reported frequencies do not directly compare the nonuniformity of learned and exact Shapley credit.

I.1 Sparse Feedback and Learned Coalition-Value Shaping

A natural objection is that the verifier already defines a coalition-value function, so one could allocate directly from re-rollout scores and dispense with the estimator. The following identifies a specific case in which exact allocation supplies equal rewards. It does not rule out informative verifier-induced credit in other games.

Proposition 2 (Equal Shapley allocation in a unanimity game)**.** *Fix a task q and a policy snapshot, and let the verifier be binary,* $\text{Verifier} \in \{0, R\}$. *Suppose the realized grand coalition succeeds while every strict subcoalition fails, i.e. $v_{q,\boldsymbol{a}}(\mathcal{N}) = R$ and $v_{q,\boldsymbol{a}}(C) = 0$ for all $C \subsetneq \mathcal{N}$. Then the induced game is the unanimity game on $\mathcal{N}$ scaled by R, and its exact Shapley allocation is*

$$\phi_i(v_{q,\boldsymbol{a}}) = R/n \quad \textit{for every } i \in \mathcal{N}.$$

On such tasks, exact verifier-induced Shapley allocation and equal allocation coincide.

The proof is immediate: the only coalition with a non-zero marginal contribution for agent i is $C = \mathcal{N} \setminus \{i\}$, whose Shapley weight is $w_{n-1} = 1/n$. **Why the restricted diagnostic does not establish equal allocation.** Consider two roles, Reasoner $= 1$ and Actor $= 2$, with

$$v(\emptyset) = v(\{1\}) = 0, \qquad v(\{2\}) = v(\{1,2\}) = 1.$$

The only answer-capable strict coalition is Actor-only, whose score matches the grand coalition, so the restricted verifier-differentiating indicator is zero. Nevertheless, exact Shapley is $(0, 1)$ and differs from equal allocation $(1/2, 1/2)$. Conversely, in a two-answer-capable-role game with $v(\emptyset) = v(\{1,2\}) = 0$ and $v(\{1\}) = v(\{2\}) = 1$, that indicator is one but exact Shapley and equal allocation both equal $(0, 0)$. Thus, the restricted-score rates cannot establish that exact Shapley equals equal allocation on the remaining tasks.

What the learned game can establish.

Training, agents	VD	LD
MATH, $n = 2$	14.20%	64.90%
MATH, $n = 3$	26.60%	76.40%
CMATH, $n = 2$	14.20%	53.50%
CMATH, $n = 3$	23.50%	67.40%
Code, $n = 2$	82.40%	82.40%
Code, $n = 3$	83.80%	83.80%

Table 20: Restricted verifier-score variation (VD) and learned allocation nonuniformity (LD). The columns measure distinct events, as defined in Appendix B. The same results appear in Table 26.

Predictions from context may be graded even when the reference verifier labels are binary, and the learned allocation can consequently differ from exact verifier-induced credit. Such a difference may reflect useful shaping, estimation error, or the change from coalition-rollout contexts at fitting time to masked grand-trajectory contexts at inference time. Table 3 compares the training variants, while the diagnostics in Appendix I describe their allocation behavior.

I.2 Estimator Fidelity

We evaluate frozen mask-conditioned predictions against verifier-induced coalition re-rollout references. Coalition-value MAE_v and RMSE_v summarize absolute and squared prediction residuals, respectively, while MAE_ϕ summarizes the error in the resulting Shapley credits. Each of the six setting-level pools contains $1{,}000$ episodes. The during-training and final-policy sets each contain $3{,}000$ episodes across the six settings. Both were collected after estimator freezing and used neither for fitting nor for selecting the freezing point.

Setting	Episodes	MAE_v	RMSE_v	MAE_ϕ	Value-error P95
MATH, $n = 2$	1000	0.054641	0.067678	0.033928	0.131633
MATH, $n = 3$	1000	0.051414	0.063803	0.022515	0.122140
CMATH, $n = 2$	1000	0.053224	0.065790	0.033715	0.125347
CMATH, $n = 3$	1000	0.051407	0.063503	0.022575	0.123624
Code, $n = 2$	1000	0.053935	0.067161	0.033703	0.129293
Code, $n = 3$	1000	0.052696	0.065297	0.022493	0.126397
During policy training	3000	0.051387	0.063873	0.026442	0.124414
Final trained policies	3000	0.053687	0.066324	0.027617	0.127066

Table 21: Estimator fidelity on post-freeze diagnostic trajectories. Setting-level rows pool the during-training and final-policy observations. P95 is the 95th percentile of absolute coalition-value prediction error. Lower error values indicate closer agreement with the verifier-defined reference game.

Setting	Seed-bootstrap 95% CI for MAE_v
MATH, $n = 2$	$[0.052700, 0.056581]$
MATH, $n = 3$	$[0.049911, 0.053546]$
CMATH, $n = 2$	$[0.052615, 0.053878]$
CMATH, $n = 3$	$[0.049452, 0.053402]$
Code, $n = 2$	$[0.051734, 0.055872]$
Code, $n = 3$	$[0.051323, 0.054175]$

Table 22: Seed-bootstrap confidence intervals accompanying Table 21. The six setting-level MAE_v values range from 0.049882 to 0.054777 during policy training and from 0.052840 to 0.054722 at the final policies. These ranges are across settings, not confidence intervals or bounds on individual prediction errors.

Interpretation. The pooled value MAE increases from 0.051387 during policy training to 0.053687 at the final policies, while credit MAE increases from 0.026442 to 0.027617. These comparisons cover two evaluated post-freeze sets, each containing 3,000 episodes. Across the six setting-level pools, value MAE ranges from 0.051407 to 0.054641, and credit MAE is lower than value MAE in every setting. The two metrics summarize different errors and do not establish a uniform bound on individual allocations.

Pooling. For setting j, let M_j denote the episode count and n_j the number of roles. Pooled value MAE weights setting-level MAEs by $M_j 2^{n_j}$, while pooled value RMSE is the square root of the similarly weighted mean of squared RMSEs. Credit MAE uses weights $M_j n_j$.

I.3 Shared-Permutation Sampling

For $n \geq 4$, the implementation uses 20 shared complete agent orders to approximate Shapley allocation of a fixed learned coalition-value game. Enumeration requires 2^n distinct coalition entries, whereas the uncompressed sampled expression contains Kn marginal terms. These counts use different units and do not determine wall-clock cost. Table 5 reports downstream performance. The analysis below characterizes sampling error conditionally on a fixed learned game.

I.3.1 Shared-permutation error for a fixed learned game

Lemma 3 (Shared-permutation allocation error)**.** *Fix $\hat{v}$ and draw K independent uniform agent permutations, using each complete permutation for every role. Let $\bar{\phi}_i$ average the sampled marginals, $\hat{\phi}_i = \phi_i(\hat{v})$, and $V_{\text{rng}} = \max_C \hat{v}(C) - \min_C \hat{v}(C)$. Then $\mathbb{E}\bar{\phi}_i = \hat{\phi}_i$ and $\sum_i \bar{\phi}_i = \hat{\Sigma}_b = \hat{v}(\mathcal{N}) - \hat{v}(\emptyset)$. For $0 < \delta < 1$, with probability at least $1 - \delta$, simultaneously for all roles,*

$$|\bar{\phi}_i - \hat{\phi}_i| \leq \alpha_K := V_{\text{rng}} \sqrt{\frac{2\log(2n/\delta)}{K}}. \tag{14}$$

Exact and sampled credits choose the same normalization branch. On the rescaling branch, $|r_i^{(K)} - r_i^{\text{exact}}| \leq R_b \alpha_K / |\hat{\Sigma}_b|$, while on fallback, both equal R_b/n.

Proof. A uniform permutation has predecessor-set probabilities equal to the Shapley weights, proving unbiasedness. The marginal contributions along each complete permutation telescope to the same

endpoint difference. Each marginal lies in $[-V_{\mathrm{rng}}, V_{\mathrm{rng}}]$, so Hoeffding's bound gives $\mathbb{P}(|\bar{\phi}_i - \hat{\phi}_i| > a) \le 2\exp[-Ka^2/(2V_{\mathrm{rng}}^2)]$ when $V_{\mathrm{rng}} > 0$. The zero-range case is exact. A union bound over roles proves Eq. (14). The shared denominator proves the branch and normalized-error statements. □

The same bounded marginals give $\mathbb{E}|\bar{\phi}_i - \hat{\phi}_i| \le V_{\mathrm{rng}}/\sqrt{K}$. These statements condition on one fixed learned game. They concern sampling error relative to its exact allocation. Under Proposition 1's separate uniform-error condition, adding β_b bounds the normalized sampled deviation from the verifier reference on the rescaling branch. Independent per-role sampling, stochastic re-evaluation of a coalition, or a changing estimator does not inherit the shared-endpoint identity. On the rescaling branch, the shared endpoint sum also gives the exact identity

$$\boldsymbol{r}^{\mathrm{samp}} - \boldsymbol{r}^{\mathrm{exact}} = \frac{R_b}{\hat{\Sigma}_b}\left(\hat{\boldsymbol{\phi}}^{\mathrm{samp}} - \hat{\boldsymbol{\phi}}^{\mathrm{exact}}\right).$$

Thus absolute errors scale by $R_b/|\hat{\Sigma}_b|$ on that branch, while on fallback, both allocations equal $R_b\mathbf{1}/n$ and their difference is zero. Any pooled ratio of rescaled to raw absolute error must use raw-error weights and account for the common fallback branch.

Empirical sampling error. Table 23 compares $K = 20$ shared-permutation estimates with exact Shapley allocation of the same learned game on 100 episodes per setting. Raw credit errors measure sampling alone, while rescaled MAE compares the resulting allocated rewards after applying the common normalization rule. The separate estimator-credit error compares exact allocation of the learned game with the verifier reference. Table 24 pools 300 episodes for each team size and reports the effect of the sampling budget. Reported raw and rescaled MAEs decrease as K increases from 8 to 32 in both pooled sets. This is an accuracy comparison and does not establish a wall-clock saving.

Setting	Raw credit MAE	Raw credit RMSE	Rescaled MAE	Estimator MAE_ϕ
MATH, $n = 4$	0.011333	0.037804	0.010655	0.016381
MATH, $n = 5$	0.013133	0.038644	0.012316	0.013270
CMATH, $n = 4$	0.012833	0.042671	0.012045	0.016472
CMATH, $n = 5$	0.010733	0.039455	0.010068	0.013153
Code, $n = 4$	0.000891	0.001504	0.000490	0.016410
Code, $n = 5$	0.000733	0.001273	0.000397	0.013265

Table 23: Sampling fidelity with $K = 20$ on 100 episodes per setting. Sampling errors use the exact learned-game allocation as their reference, while estimator error uses the verifier-induced game.

Agents	K	Raw credit MAE	Raw credit RMSE	Rescaled MAE
4	8	0.014597	0.054741	0.013506
4	16	0.011225	0.042246	0.010418
4	20	0.008353	0.032925	0.007730
4	32	0.007879	0.029377	0.007310
5	8	0.011926	0.050428	0.011040
5	16	0.008588	0.035366	0.007942
5	20	0.008200	0.031894	0.007594
5	32	0.005816	0.024105	0.005386

Table 24: Sampling-budget comparison on 300 pooled episodes per team size. Both raw credit errors and rescaled reward errors are relative to exact allocation of the same learned game.

I.4 Internal Estimator Variants

We further analyze internal design choices of the learned LLM-SHAP estimator. While Table 3 evaluates algorithm-level coalition-value acquisition choices, this diagnostic focuses on whether different LLM-SHAP variants produce accurate coalition-value estimates and faithful Shapley allocations.

Protocol. We compute this fixed-trajectory comparison on the pooled MATH, $n = 2$ diagnostic set of 1,000 logged grand-coalition trajectories and verifier-derived reference labels, without re-running reinforcement fine-tuning for each variant. Its Frozen row matches the corresponding row of Table 21. For each rollout step b and coalition $C \subseteq \mathcal{N}$, each variant produces an allocation-time coalition-value prediction $\hat{v}(C)$ from the same mask-conditioned context $x_{b,C}^{\mathrm{infer}}$. We compare $\hat{v}(C)$ with the

Variant	$\text{MAE}_v \downarrow$	$\text{RMSE}_v \downarrow$	$\text{MAE}_\phi \downarrow$
Frozen LLM-SHAP	0.054641	0.067678	0.033928
Hard-zero infer.	0.034327	0.057445	0.037613
Same-step fit	0.058803	0.072741	0.039314
Delayed checkpoint	0.063122	0.078936	0.046743

Table 25: Internal diagnostics on the pooled MATH, $n = 2$ set. "Same-step fit" denotes fitting on current-step labels before prediction and is reported only as an estimator diagnostic, not as the main allocation protocol. "Hard-zero infer." clamps no-answer-capable coalitions to zero at inference time. Lower values indicate better coalition-value prediction and more faithful Shapley allocation.

verifier-induced coalition value $v_b^{\text{env}}(C)$, and compare the resulting Shapley credits $\hat{\phi}_{b,i}$ with the verifier-induced Shapley credits $\phi_{b,i}^{\text{env}}$.

Frozen LLM-SHAP is the default learned allocation protocol used in Algorithm 1. It updates the estimator only before the estimator-freezing rule is satisfied and then uses the frozen estimator for mask-conditioned allocation-time coalition-value prediction. **Same-step fit** fits the value head on the current diagnostic labels before predicting coalition values. It is an internal diagnostic, not the main training protocol. **Delayed checkpoint** evaluates a lagged value-head checkpoint on the same diagnostic inputs. **Hard-zero inference** clamps coalitions without answer-capable agents to zero at inference time before applying the Shapley formula.

Hard-zero decomposition. Let Z be the no-answer-capable coalition entries clamped to zero, with zero verifier reference values, and let p_Z be their fraction under the common diagnostic weights. Holding all other predictions and weights fixed gives

$$\text{MAE}_v^{\text{hz}} = \text{MAE}_v^{\text{fr}} - p_Z \mathbb{E}_Z|\hat{v}|, \qquad (\text{RMSE}_v^{\text{hz}})^2 = (\text{RMSE}_v^{\text{fr}})^2 - p_Z \mathbb{E}_Z \hat{v}^2.$$

For the equal-weight coalition diagnostic with one answer-capable role, $p_Z = 1/2$. On this diagnostic set, the changes from 0.054641 to 0.034327 in MAE_v and from 0.067678 to 0.057445 in RMSE_v obey these non-increase conditions. Shapley-credit error need not decrease because the clamped entries change coalition marginal contributions.

Interpretation. The same-step fitted diagnostic tests whether fitting on the current labels before prediction changes estimator behavior. It is not the allocation protocol. The delayed-checkpoint diagnostic has higher reported errors than Frozen LLM-SHAP in Table 25. On this pooled MATH two-agent set, hard-zero inference lowers value errors but increases MAE_ϕ from 0.033928 to 0.037613. Modifying coalition entries changes the marginal contributions used by Shapley allocation.

I.5 Allocation Stability and Differentiating Rate

We next evaluate whether the learned coalition-value estimator produces stable and role-differentiated allocation-time rewards under sparse verifier feedback after subcoalition rollouts have been stopped. These diagnostics describe the magnitudes and nonuniformity of rewards obtained from frozen masked-context predictions.

Episode-level interpretation. Let $\mathcal{D}$ denote the post-freeze episode set used for a diagnostic. For a failed task, Eq. (4) returns zero reward for all roles, so its learned differentiating indicator is zero. On a common episode set, the learned differentiating rate is consequently a lower bound on the fraction with positive reward. The role-share averages in Appendix I.6 are explicitly conditioned on successful episodes. These conditional averages and the task-level error summaries describe different quantities and should not be interpreted as interchangeable measures of fidelity.

Diagnostic metrics. On the common post-freeze diagnostic set $\mathcal{D}$, the sum of raw learned Shapley credits is

$$\hat{\Sigma}_b = \sum_i \hat{\phi}_{b,i} = \hat{v}_b(\mathcal{N}) - \hat{v}_b(\emptyset).$$

We report its absolute mean and fifth percentile. The magnitude-trigger rate is

$$\frac{1}{|\mathcal{D}|} \sum_{b \in \mathcal{D}} \mathbf{1}\{|\hat{\Sigma}_b| < \tau_{\text{floor}}\}, \qquad \tau_{\text{floor}} = 0.05 R_{\max}.$$

This statistic concerns the denominator magnitude only. The rescaling rule also checks sign agreement. In particular, a zero-reward episode can fail the sign condition even when the magnitude exceeds the

Setting	Mean $\lvert\hat{\Sigma}\rvert$	P5 $\lvert\hat{\Sigma}\rvert$	Magnitude trigger rate	$\Pr(\lvert\hat{\Sigma}\rvert < 0.08)$	MAE_v	VD	LD
MATH, $n = 2$	0.858856	0.063994	0.00%	20.90%	0.054641	14.20%	64.90%
MATH, $n = 3$	0.876545	0.063667	0.00%	19.10%	0.051414	26.60%	76.40%
CMATH, $n = 2$	0.744431	0.062985	0.00%	32.30%	0.053224	14.20%	53.50%
CMATH, $n = 3$	0.780726	0.063571	0.00%	28.70%	0.051407	23.50%	67.40%
Code, $n = 2$	0.128318	0.064421	0.00%	17.60%	0.053935	82.40%	82.40%
Code, $n = 3$	0.129423	0.065293	0.00%	16.20%	0.052696	83.80%	83.80%

Table 26: Allocation-stability diagnostics on 1,000 pooled episodes per setting. The magnitude-trigger column concerns only $\lvert\hat{\Sigma}\rvert < \tau_{\text{floor}}$, not the full sign-aware fallback rule. VD and LD retain their distinct event definitions.

Setting	Pooled mean R_b	$\Pr(R_b > 0)$	During-training mean R_b
MATH, $n = 2$	0.791000	79.10%	0.792000
MATH, $n = 3$	0.809000	80.90%	0.816000
CMATH, $n = 2$	0.677000	67.70%	0.688000
CMATH, $n = 3$	0.713000	71.30%	0.738000
Code, $n = 2$	0.060912	82.40%	0.060195
Code, $n = 3$	0.061944	83.80%	0.061192

Table 27: Reward summaries for the allocation diagnostic sets. For binary mathematics rewards, mean reward equals the positive-reward fraction. For code, R_b is the fraction of unit tests passed, so its mean differs from the probability of any positive reward. Diagnostic-set rewards are not additional held-out benchmark accuracy results.

threshold. Zero magnitude-trigger rates must not be read as zero full-fallback rates. The additional probability $\Pr(\lvert\hat{\Sigma}_b\rvert < 0.08)$ is a diagnostic at a different threshold, not the implemented safeguard rate ($R_{\max} = 1$ here).

Coalition-value error uses equal weight for all coalition entries within each episode:

$$\text{MAE}_v = \frac{1}{|\mathcal{D}|2^n} \sum_{b \in \mathcal{D}} \sum_{C \subseteq \mathcal{N}} |\hat{v}_b(C) - v_b^{\text{env}}(C)|.$$

Its setting-level values match Table 21. Verifier-Diff. (VD) retains the restricted event defined in Appendix B: an answer-capable nonempty strict coalition has a reference score different from the grand coalition. This is not a test of equality of exact Shapley credits. Learned differentiation (LD) uses

$$\delta_b := \frac{\|\boldsymbol{r}_b^{\text{LLM-SHAP}} - R_b \mathbf{1}/n\|_1}{R_b + \epsilon} > 0.1.$$

Interpretation. The fifth percentiles of $\lvert\hat{\Sigma}\rvert$ lie between 0.062985 and 0.065293. The magnitude-trigger rate is 0.00% in all six sets, while 16.20%–32.30% of episodes lie below the higher diagnostic threshold of 0.08.

Across mathematics settings, restricted verifier-score variation (VD) ranges from 14.20% to 26.60%, while learned allocation nonuniformity (LD) ranges from 53.50% to 76.40%. For code, both rates are 82.40% with two agents and 83.80% with three agents. Among positive-reward three-agent mathematics episodes, 5.56% on MATH and 5.47% on CMATH receive near-equal learned allocations (Table 28). VD and LD measure different events, so equality of their rates does not establish equality of their episode-level indicators or credit vectors.

I.6 Role-Wise Credit Distribution

We summarize allocated reward shares conditional on positive team reward. The share of role i is $r_{b,i}^{\text{LLM-SHAP}}/R_b$ for $R_b > 0$. The near-equal column reports $\Pr(\delta_b \le 0.1 \mid R_b > 0)$ using the same normalized deviation as the LD diagnostic. The answer-share column measures the fraction of positive episodes in which the answer-producing role receives more than 0.9 of the team reward.

Setting	Positive episodes	Role 1	Role 2	Role 3	Near-equal ($\delta_b \leq 0.1$)	Answer share > 0.9
MATH, $n = 2$	791	0.096732	**0.903268**	—	17.95%	82.05%
MATH, $n = 3$	809	0.109586	0.068932	**0.821482**	5.56%	67.12%
CMATH, $n = 2$	677	0.111070	**0.888930**	—	20.97%	79.03%
CMATH, $n = 3$	713	0.096972	0.081972	**0.821056**	5.47%	67.04%
Code, $n = 2$	824	0.133826	**0.866174**	—	0.00%	10.92%
Code, $n = 3$	838	0.125468	0.084040	**0.790492**	0.00%	0.00%

Table 28: Mean reward shares on positive-reward diagnostic episodes. For mathematics, the role order is Reasoner–Actor with two agents and Reasoner–Actor–Reviewer with three agents. For code, the order is Reasoner–Coder with two agents and Reasoner–Coder–Reviewer with three agents. Bold entries identify the designated final-answer role. Dashes indicate that no third role exists. The corresponding normalized fixed mean-share vectors are used by the Role prior ablation in Table 3. Displayed means may differ from a unit sum in the last decimal because of rounding.

Interpretation. The designated final-answer role has the largest mean reward share in all six settings. In three-agent mathematics, the Reviewer receives mean shares of 0.821482 on MATH and 0.821056 on CMATH, while in three-agent code, its mean share is 0.790492. In two-agent mathematics, the Actor receives more than 0.9 of the team reward in 82.05% of positive MATH episodes and 79.03% of positive CMATH episodes. The corresponding fraction for the Coder in two-agent code is 10.92%. These statistics describe learned reward allocation conditional on positive team reward. They do not establish ground-truth causal contributions. Appendix I.2 compares the learned estimates with the reference coalition game.

The near-equal fractions satisfy

$$\Pr(\delta_b \leq 0.1 \mid R_b > 0) = 1 - \frac{\Pr(\delta_b > 0.1)}{\Pr(R_b > 0)},$$

because the implemented rule assigns zero credit on failed tasks.

J LIMITATIONS

Evaluation covers fixed roles, 3B models, greedy decoding, and task-level rewards. Frozen prediction must transfer to masked contexts under policy drift. Causal attribution depends on matched controls. Resource-matched results support descriptive mean comparisons. Failed tasks receive zero rewards but potentially nonzero critic-subtracted advantages. Sampling and generalization beyond the tested teams remain open.

Evaluation scope. The experiments cover mathematical reasoning and competitive programming with 3B-scale backbones, fixed role protocols, and task-level terminal rewards. MASHAP-LLM training uses greedy decoding with one full-team trajectory per problem. The results do not establish MASHAP-LLM training behavior under stochastic exploration, open-ended dialogue, persistent-state interaction, or heterogeneous backbone assignments. Different dataset sizes and training intensities also limit comparisons between the MATH- and CMATH-trained regimes.

Computation and larger teams. Exact coalition enumeration is exponential in the number of agents. Shared-permutation sampling changes this workload but introduces approximation error, and the four- and five-agent settings do not determine how to select the sampling budget for substantially larger teams. Early supervision requires coalition re-rollouts and estimator fitting, while frozen allocation retains estimator inference. In the matched measured setting, full-run training takes $1.473\times$ the single-agent PPO baseline time, versus $1.563\times$ for verifier-induced allocation. These relative costs do not establish a speedup across other hardware, tasks, or training budgets.

Estimator and verifier dependence. The estimator is fitted on re-rollout contexts and subsequently applied to masked full-team trajectories while policies continue to change. The reliability results cover the drift and random label corruption in the reported settings, not arbitrary policy shifts, systematic verifier bias, or domains without a programmatic verifier. The allocation properties in Appendix C are conditional statements about a reference game or a fixed batch. Empirical MAE does not verify a uniform error bound over all coalitions. The complete-pipeline comparisons do not isolate components. The paired component controls address several of these alternatives, with their

interpretation restricted to the stated interventions and endpoints. Reward nonuniformity has not been validated against ground-truth causal contributions, and aggregate error measures do not establish per-episode role ordering.

Credit on failed tasks. Under Eq. (4), a failed task with $R_b = 0$ gives zero allocated reward to every role. The current rule therefore cannot assign blame among roles on failures, even though signed Shapley credits could sum to zero while remaining individually nonzero. Signed failure credit and refresh schedules beyond the tested midpoint update remain directions for future work.

K Paired Replication, Resource-Matched Comparisons, and Component Controls

This appendix reports the protocols and complete results for fixed-update comparisons across team sizes, resource-matched comparisons, and component and schedule controls. Table 4 summarizes the MATH500 and Codeforces columns of the two resource-matched performance tables. The six-benchmark scores and resource ledger appear below. The main performance comparisons in Tables 1–2 use five seeds. The paired comparisons below use separate 24/12-seed mathematical/code sets, with means, SDs, and intervals computed within each seed set.

K.1 Paired Seeds, Endpoints, and Prespecified Inference

Seed sets and scoring. Mathematical comparisons use $m = 24$ paired seeds per training setting: $103, 113, \ldots, 333$ for MATH and $503, 513, \ldots, 733$ for CMATH. Codeforces comparisons use $m = 12$ paired seeds, $1003, 1013, \ldots, 1113$. Seed identifiers advance by ten. Pairing is within a training setting and comparison, not between the MATH and CMATH seed sets. Mathematical accuracy is $100\,\mathrm{correct}/N$, while code uses the mean unit-test fraction. The analysis protocol retains 17 significant digits before aggregation and rounds only displayed summaries. The full method uses the same seed records across component comparisons and all associated test benchmarks. Consequently, these comparisons and the related benchmarks are dependent. Bonferroni control does not require their independence.

Superiority family. The prespecified two-sided superiority family has $K_{\mathrm{sup}} = 194$ comparisons. Fixed-update comparisons contribute 80: six team-size/training-domain groups with twelve prespecified baseline/benchmark contrasts per group ($6 \times 12 = 72$), plus eight four/five-agent comparisons against C3. The twelve contrasts per group count hypotheses, not necessarily twelve distinct methods. Resource-matched comparisons contribute 72: two resource envelopes, each with two domains, six baselines, and three benchmarks ($2 \times 2 \times 6 \times 3$). Task–credit alignment, coalition-value supervision, credit aggregation, and reward transformation controls contribute 42: respectively $6/18/12/6$ comparisons over the six benchmarks. Raw-credit and learning-rate/floor sensitivity rows are descriptive rather than additional confirmatory tests in this family. The fixed-update and component tables display selected endpoints, and the reported superiority conclusions apply to the corresponding contrasts.

For paired differences d_s with mean $\bar{d}$ and sample SD s_d (divisor $m - 1$), the superiority interval is

$$\bar{d} \pm t_{m-1,\,1-0.05/(2K_{\mathrm{sup}})} \frac{s_d}{\sqrt{m}}. \tag{15}$$

Positive lower bounds support superiority for the corresponding contrast. The normal-difference working model is the basis of the paired t calculation. Descriptive averages over benchmark cells have no implied confidence interval.

Noninferiority and costs. A separate, prespecified one-sided family contains four comparisons: allocation inputs and estimator freezing/refresh at MATH500 and Codeforces. With $K_{\mathrm{NI}} = 4$, the lower bound is

$$L = \bar{d} - t_{m-1,\,1-0.05/K_{\mathrm{NI}}} \frac{s_d}{\sqrt{m}}. \tag{16}$$

The margins, selected before viewing the results, are $\delta = 1.0$ percentage point for MATH500 and 0.5 for Codeforces. Noninferiority requires $L > -\delta$. Failure to find superiority is not used as evidence of equivalence. The two cost intervals form their own $K_{\mathrm{cost}} = 2$ family for percentage GPU-hour saving. These are separately controlled families, each at family-wise level 0.05. No joint 0.05 error claim is made across all three families. Per-seed cost percentages, rather than a ratio of aggregate mean costs, define the cost endpoint.

K.2 Fixed-Update Comparisons across Team Sizes

Agents	Train / test	Comparator	Full	Comparator score	Difference [CI]
2	MATH / MATH500	C3	52.08 ± 0.44	48.70 ± 0.86	3.38 $[2.62, 4.13]$
2	MATH / CMATH	CCPO	84.87 ± 0.93	82.51 ± 0.93	2.36 $[1.35, 3.38]$
2	MATH / GSM8K	C3	80.50 ± 0.81	78.60 ± 1.00	1.89 $[0.94, 2.84]$
2	CMATH / MATH500	C3	52.73 ± 0.72	49.23 ± 0.36	3.51 $[2.88, 4.14]$
2	CMATH / CMATH	C3	84.05 ± 0.76	82.36 ± 0.65	1.69 $[0.86, 2.52]$
2	CMATH / GSM8K	CCPO	81.04 ± 0.97	77.60 ± 0.82	3.44 $[2.56, 4.32]$
2	Code / Codeforces	C3	5.75 ± 0.17	4.68 ± 0.33	1.06 $[0.54, 1.58]$
2	Code / MBPP-test	C3	9.69 ± 0.82	6.96 ± 1.15	2.73 $[0.91, 4.55]$
2	Code / MBPP+	C3	9.18 ± 0.80	6.88 ± 0.63	2.30 $[1.24, 3.36]$
3	MATH / MATH500	C3	54.23 ± 1.01	48.19 ± 1.28	6.04 $[4.87, 7.21]$
3	MATH / CMATH	Equal	85.84 ± 0.89	80.96 ± 1.07	4.88 $[3.76, 6.01]$
3	MATH / GSM8K	C3	82.34 ± 0.85	78.65 ± 0.88	3.69 $[2.90, 4.49]$
3	CMATH / MATH500	C3	54.98 ± 0.48	47.70 ± 0.62	7.28 $[6.60, 7.97]$
3	CMATH / CMATH	C3	86.13 ± 1.62	83.64 ± 1.06	2.49 $[1.10, 3.87]$
3	CMATH / GSM8K	C3	82.55 ± 0.90	78.14 ± 0.72	4.41 $[3.54, 5.27]$
3	Code / Codeforces	C3	6.46 ± 0.34	4.94 ± 0.23	1.52 $[1.08, 1.96]$
3	Code / MBPP-test	C3	19.67 ± 1.84	15.15 ± 1.70	4.52 $[2.17, 6.86]$
3	Code / MBPP+	C3	18.87 ± 1.36	14.74 ± 1.45	4.13 $[1.93, 6.33]$

Table 29: Fixed-update paired comparisons with two and three agents. Values are means $\pm$ sample SD, and differences and intervals are in percentage points. Mathematical comparisons use 24 paired seeds and code uses 12. Intervals use the two-sided Bonferroni superiority family, $K_{\mathrm{sup}} = 194$. Each row compares the full method against the named baseline.

Agents	Test	Full	C3	Difference [CI]
4	MATH500	55.07 ± 0.76	51.48 ± 1.05	3.58 $[2.78, 4.39]$
4	CMATH	85.46 ± 0.98	81.10 ± 0.98	4.36 $[3.61, 5.10]$
4	GSM8K	82.62 ± 0.69	77.99 ± 0.90	4.63 $[3.72, 5.54]$
4	Codeforces	6.36 ± 0.37	4.99 ± 0.29	1.37 $[0.82, 1.92]$
5	MATH500	54.32 ± 0.73	50.66 ± 1.00	3.66 $[2.66, 4.65]$
5	CMATH	85.88 ± 0.89	80.86 ± 1.19	5.02 $[4.10, 5.95]$
5	GSM8K	82.33 ± 0.81	77.35 ± 1.02	4.98 $[4.07, 5.88]$
5	Codeforces	6.45 ± 0.35	4.92 ± 0.35	1.53 $[0.94, 2.13]$

Table 30: Fixed-update paired comparisons with four and five agents against C3. Mathematical tasks use MATH training. Intervals use $K_{\mathrm{sup}} = 194$. Role composition also changes with team size.

Interpretation. All 26 displayed fixed-update differences and corrected intervals are positive. From the rounded displayed differences, the two-/three-agent nine-setting averages are approximately $2.48/4.33$ points, and the four-/five-agent four-setting averages are approximately $3.49/3.80$ points. These are margins over the named comparators. The three-agent MATH-to-CMATH row compares against Equal. The larger-team experiment changes roles together with team size and therefore evaluates the resulting team configurations.

K.3 Complete Resource-Matched Comparisons

Tables 31 and 32 give the complete results behind the main-text summary (Table 4). Table 33 reports the corresponding resource usage. All comparisons use three agents ($n = 3$), with the same backbone, role topology, training data, and evaluation settings within each domain as the three-agent fixed-update comparisons. The two panels impose GPU-hour and generated-output-token caps separately and evaluate the final checkpoint permitted by the corresponding cap, whereas fixed-update comparisons select checkpoints by training-update count. C3, CCPO, SEPO, SHARP, AT-GRPO, and Shared-GRPO are the six baselines. SEPO denotes the self-/peer-evaluation credit variant described in Li et al. (2026b). AT-GRPO is the agent-/turn-wise optimization baseline of Zhao et al. (2026). Methods retain their native sampling and optimization mechanisms, so a policy update is not a common unit of task exposure. Verifier calls are recorded separately. The token envelope measures generated output tokens. Prompt prefill and other model forward passes lie outside this matching criterion.

Method	MATH500	CMATH	GSM8K	Codeforces	MBPP-test	MBPP+
MASHAP	54.36 ± 1.02	85.79 ± 0.94	82.43 ± 0.89	6.50 ± 0.36	19.53 ± 1.83	18.96 ± 1.32
C3	52.34 ± 1.27	84.58 ± 0.97	80.96 ± 0.93	5.72 ± 0.34	17.44 ± 1.88	17.04 ± 1.39
CCPO	51.59 ± 0.96	84.49 ± 0.92	80.54 ± 1.04	5.57 ± 0.48	16.72 ± 2.03	16.59 ± 1.54
SEPO	52.57 ± 1.10	84.90 ± 0.85	81.26 ± 0.81	5.86 ± 0.35	17.70 ± 1.97	17.35 ± 1.33
SHARP	51.45 ± 0.98	84.10 ± 0.91	80.23 ± 1.10	5.50 ± 0.37	16.47 ± 1.88	16.27 ± 1.21
AT-GRPO	52.47 ± 1.10	84.56 ± 1.29	81.29 ± 1.08	5.80 ± 0.38	17.55 ± 1.85	17.40 ± 1.27
Shared-GRPO	51.19 ± 1.04	83.87 ± 1.00	80.05 ± 1.05	5.31 ± 0.30	15.88 ± 1.64	15.68 ± 1.03

Table 31: GPU-hour-envelope comparison with three agents ($n = 3$). Entries are means $\pm$ sample SD from the final checkpoints permitted by the GPU-hour cap, using 24 seeds for mathematics and 12 for code. Mathematics uses MATH training, and code uses Codeforces training.

Method	MATH500	CMATH	GSM8K	Codeforces	MBPP-test	MBPP+
MASHAP	54.56 ± 0.98	86.00 ± 0.89	82.25 ± 0.87	6.58 ± 0.34	19.91 ± 1.84	19.00 ± 1.37
C3	52.27 ± 1.12	84.84 ± 1.13	80.90 ± 1.04	5.95 ± 0.38	17.81 ± 1.88	16.97 ± 1.26
CCPO	51.98 ± 1.14	84.66 ± 0.85	80.23 ± 1.05	5.71 ± 0.40	17.20 ± 1.85	16.53 ± 1.40
SEPO	52.83 ± 1.04	85.06 ± 1.08	81.04 ± 1.01	5.87 ± 0.33	18.22 ± 1.89	17.32 ± 1.21
SHARP	51.57 ± 1.18	84.29 ± 0.98	80.19 ± 1.04	5.44 ± 0.37	16.95 ± 1.98	16.24 ± 1.31
AT-GRPO	52.78 ± 1.07	84.63 ± 1.00	80.94 ± 0.76	5.86 ± 0.33	18.03 ± 1.62	17.51 ± 1.55
Shared-GRPO	51.46 ± 1.30	84.21 ± 1.13	80.08 ± 0.98	5.46 ± 0.41	16.39 ± 2.17	15.88 ± 1.26

Table 32: Generated-output-token-envelope comparison with three agents ($n = 3$). Entries are means $\pm$ sample SD from the final checkpoints permitted by the output-token cap, using 24 seeds for mathematics and 12 for code. Mathematics uses MATH training, and code uses Codeforces training.

Interpretation. Across the six columns, GPU-envelope margins over the strongest competitor are $1.79, 0.89, 1.14, 0.64, 1.83, 1.56$ points, and token-envelope margins are $1.73, 0.94, 1.21, 0.63, 1.69, 1.49$. Thus all twelve descriptive mean comparisons favor MASHAP-LLM. The GPU-hour ranges within a domain differ by less than 0.4% of their means, and output-token ranges differ by less than 0.21%. These results compare performance under approximately matched GPU-hour and generated-output-token budgets, respectively.

K.4 Component Interventions

The common control is the full three-agent pipeline, with MATH training for mathematical tasks and Codeforces training for code. The task–credit alignment control changes task-to-credit correspondence while preserving the conditional share distribution. Coalition-value supervision controls change coalition-label information using label shuffling, grand-coalition-only supervision, and an oracle-rerolled reference. The allocation-input control replaces masked full-team contexts with coalition rerollouts. Credit aggregation controls hold coalition information fixed and change its aggregation to LOO or Additive-role. Reward transformation controls use additive centering or raw credits. The declared reward conventions remain fixed. All comparisons reuse the full-method seed records rather than treating repeated appearances as independent runs.

Oracle reference. For Oracle-rerolled only, the paired left-hand method is Learned-rerolled (the Rerolled row), not the masked Full row. This holds the allocation-input construction fixed: the reported mean differences are $54.47 - 53.02 = 1.45$ on MATH500 and $6.51 - 5.98 = 0.53$ on Codeforces. Full-versus-oracle would change the input construction as well. Rounding can change other displayed subtractions by 0.01.

Task alignment and supervision. Task-shuffled allocation loses $2.08/0.63$ points on MATH500/Codeforces, and its mean losses are also positive on the remaining four benchmarks. Label-shuffled and grand-only supervision likewise have positive primary-endpoint differences with corrected lower bounds above zero. These controls support useful task-conditioned coalition information under the specified matching protocol. Learned-rerolled also exceeds the oracle-rerolled reference, which supports its usefulness for policy optimization rather than greater accuracy than the verifier at defining the reference game.

Aggregation and transformation. LOO and Additive-role have positive corrected losses at both primary endpoints, supporting the Shapley operator against these same-information alternatives. Additive centering loses $1.00/0.44$ points at MATH500/Codeforces with positive corrected lower bounds.

Envelope/domain	Method	GPU-h	Output B tokens	Verifier M calls	Updates
gpu/MATH	MASHAP	2380.8	5.223	2.279	6481
gpu/MATH	C3	2384.3	5.024	3.563	617
gpu/MATH	CCPO	2379.9	4.827	2.357	678
gpu/MATH	SEPO	2381.2	4.607	1.867	672
gpu/MATH	SHARP	2383.4	4.390	3.312	522
gpu/MATH	AT-GRPO	2379.0	4.160	1.540	752
gpu/MATH	Shared-GRPO	2386.2	3.951	1.462	714
gpu/Code	MASHAP	3182.7	3.920	1.367	3863
gpu/Code	C3	3176.6	3.738	2.040	356
gpu/Code	CCPO	3170.9	3.608	1.399	403
gpu/Code	SEPO	3173.1	3.437	1.101	397
gpu/Code	SHARP	3176.5	3.274	1.972	307
gpu/Code	AT-GRPO	3175.8	3.128	0.919	449
gpu/Code	Shared-GRPO	3170.5	2.943	0.865	422
token/MATH	MASHAP	2510.5	5.973	2.622	7431
token/MATH	C3	2610.8	5.973	4.255	736
token/MATH	CCPO	2705.3	5.977	2.917	841
token/MATH	SEPO	2841.2	5.978	2.409	871
token/MATH	SHARP	2966.6	5.974	4.585	716
token/MATH	AT-GRPO	3059.4	5.973	2.211	1080
token/MATH	Shared-GRPO	3155.6	5.973	2.211	1080
token/Code	MASHAP	3283.7	4.484	1.569	4438
token/Code	C3	3446.7	4.482	2.527	437
token/Code	CCPO	3604.7	4.479	1.749	503
token/Code	SEPO	3783.8	4.475	1.439	520
token/Code	SHARP	3902.3	4.478	2.633	419
token/Code	AT-GRPO	4136.7	4.482	1.317	643
token/Code	Shared-GRPO	4235.1	4.479	1.317	643

Table 33: Resource usage for the three-agent budget comparisons. GPU-h denotes accumulated GPU-hours, B billion generated output tokens, and M million verifier calls. Updates retain each method's own unit. Each envelope matches its specified resource separately.

Variant	MATH500	CMATH	GSM8K
MASHAP	54.23 ± 1.01	85.84 ± 0.89	82.34 ± 0.85
Task-shuffled	52.15 ± 1.73	84.93 ± 1.11	81.18 ± 1.15
Label-shuffled	51.56 ± 1.49	84.35 ± 1.29	80.79 ± 1.28
Grand-only	52.33 ± 1.22	84.70 ± 0.96	81.05 ± 0.99
LOO	52.88 ± 1.13	85.02 ± 0.91	81.45 ± 0.99
Additive-role	52.57 ± 1.36	84.76 ± 1.10	81.27 ± 1.14
Additive-centering	53.23 ± 1.06	85.06 ± 1.09	81.42 ± 1.01
Raw-credit	53.80 ± 1.37	85.96 ± 1.19	82.08 ± 1.23
Rerolled	54.47 ± 1.13	85.92 ± 0.99	82.27 ± 1.02
Refresh	54.54 ± 1.21	85.88 ± 1.04	82.33 ± 1.04
Half-head-LR	54.03 ± 1.19	85.85 ± 1.18	82.16 ± 1.20
Double-head-LR	54.17 ± 1.28	85.72 ± 1.12	82.36 ± 0.98
Cosine-policy	54.44 ± 1.48	85.79 ± 1.15	82.44 ± 1.18
Floor-0.025	54.06 ± 1.01	85.92 ± 0.98	82.32 ± 0.94
Floor-0.10	53.92 ± 1.29	85.87 ± 1.07	82.20 ± 1.09
Oracle-rerolled	53.02 ± 1.50	84.96 ± 1.32	81.28 ± 1.30

Table 34: Component and sensitivity results: MATH training, 24 paired seeds, mean $\pm$ sample SD. All variants reuse the same full-method seed records. Sensitivity rows and raw-credit results are descriptive and are not included in the 42-comparison component superiority allocation.

Variant	Codeforces	MBPP-test	MBPP+
MASHAP	6.46 ± 0.34	19.67 ± 1.84	18.87 ± 1.36
Task-shuffled	5.83 ± 0.46	18.03 ± 1.84	17.39 ± 1.44
Label-shuffled	5.66 ± 0.33	17.47 ± 1.79	16.91 ± 1.73
Grand-only	5.90 ± 0.41	18.08 ± 1.88	17.40 ± 1.24
LOO	5.98 ± 0.39	18.39 ± 1.69	17.70 ± 1.07
Additive-role	5.85 ± 0.40	18.10 ± 1.99	17.50 ± 1.45
Additive-centering	6.02 ± 0.45	18.49 ± 1.79	17.81 ± 1.28
Raw-credit	6.27 ± 0.47	19.40 ± 2.04	19.00 ± 1.96
Rerolled	6.51 ± 0.39	19.78 ± 2.01	18.81 ± 1.33
Refresh	6.52 ± 0.38	19.84 ± 1.69	18.95 ± 1.55
Half-head-LR	6.40 ± 0.42	19.34 ± 1.89	18.73 ± 1.43
Double-head-LR	6.39 ± 0.45	19.74 ± 2.31	18.66 ± 1.48
Cosine-policy	6.49 ± 0.35	19.52 ± 2.04	19.05 ± 1.71
Floor-0.025	6.40 ± 0.43	19.56 ± 1.78	18.98 ± 1.28
Floor-0.10	6.39 ± 0.45	19.72 ± 2.04	18.75 ± 1.35
Oracle-rerolled	5.98 ± 0.53	18.35 ± 2.28	17.59 ± 1.48

Table 35: Component and sensitivity results: Codeforces training, 12 paired seeds, mean $\pm$ sample SD. The same checkpoints are evaluated on the three code benchmarks. MBPP-test and MBPP+ share task ancestry.

Comparator	Endpoint	Paired SD	Difference [CI]	Negative seeds
Task-shuffled	MATH500	1.08	$2.08\ [1.13, 3.04]$	0
Task-shuffled	Codeforces	0.25	$0.63\ [0.24, 1.02]$	0
Label-shuffled	MATH500	1.18	$2.68\ [1.64, 3.71]$	0
Label-shuffled	Codeforces	0.28	$0.80\ [0.37, 1.23]$	0
Grand-only	MATH500	0.95	$1.90\ [1.06, 2.74]$	0
Grand-only	Codeforces	0.24	$0.56\ [0.20, 0.93]$	0
Oracle-rerolled	MATH500	0.87	$1.45\ [0.69, 2.21]$	1
Oracle-rerolled	Codeforces	0.21	$0.53\ [0.21, 0.85]$	0
LOO	MATH500	0.93	$1.36\ [0.54, 2.18]$	0
LOO	Codeforces	0.18	$0.49\ [0.21, 0.76]$	0
Additive-role	MATH500	0.85	$1.67\ [0.92, 2.41]$	1
Additive-role	Codeforces	0.21	$0.61\ [0.30, 0.93]$	0
Additive-centering	MATH500	0.80	$1.00\ [0.30, 1.70]$	2
Additive-centering	Codeforces	0.15	$0.44\ [0.21, 0.68]$	0

Table 36: Primary-endpoint paired superiority comparisons, using $K_{\text{sup}} = 194$. Paired SD is the sample SD of seed differences. Positive differences favor the full method, except that Oracle-rerolled compares learned-rerolled against oracle-rerolled to hold allocation inputs fixed. Negative seeds count differences below zero.

However, raw credits have slightly higher means on CMATH (85.96 versus 85.84) and MBPP+ (19.00 versus 18.87), so no claim of uniform superiority over raw credits follows. The transformation's budget identity is mathematical, while its optimization benefits are empirical and comparator-dependent. The sensitivity rows describe nearby settings but have no additional confirmatory significance claim.

K.5 Allocation Inputs and Estimator Freezing/Refresh

The frozen and refreshed branches start from the same early-fit checkpoint with identical value-head parameters and a shared early-stage cost. Each refreshed seed undergoes one post-freeze refit at the intermediate checkpoint between freezing and final evaluation. Audit-only work is reported separately. The allocation-input comparison evaluates masked versus rerolled inputs, while the freezing comparison evaluates frozen versus refreshed heads. Performance endpoints are paired within the relevant seed sets.

Comparator	Endpoint	Margin	Difference	Paired SD	Lower bound	NI
Rerolled	MATH500	1.0	-0.23	0.52	-0.49	Yes
Rerolled	Codeforces	0.5	-0.05	0.16	-0.17	Yes
Refresh	MATH500	1.0	-0.31	0.59	-0.60	Yes
Refresh	Codeforces	0.5	-0.05	0.18	-0.19	Yes

Table 37: Prespecified noninferiority (NI) comparisons, with margins in percentage points and a one-sided $K_{\mathrm{NI}} = 4$ Bonferroni family. Differences are masked/frozen minus rerolled/refreshed. NI requires the reported lower bound to exceed the negative margin.

Domain	Frozen GPU-h	Refresh GPU-h	Full saving (%) [CI]	Post saving (%)	Audit-only GPU-h
MATH	2324.2	2590.1	10.24 $[9.54, 10.95]$	12.35	746.4
Code	3052.7	3424.0	10.83 $[9.84, 11.81]$	12.98	975.6

Table 38: Frozen versus refreshed estimator costs. Full-run saving is computed per seed as $100(C_{\mathrm{refresh}} - C_{\mathrm{frozen}})/C_{\mathrm{refresh}}$, with a separate two-domain cost family, $K_{\mathrm{cost}} = 2$. Audit-only work is listed separately from deployable training cost. GPU-hours measure accumulated device time.

Interpretation and scope. The four reported lower bounds exceed the prespecified negative margins, supporting noninferiority within $1.0/0.5$ points at the two primary endpoints. The frozen-versus-refresh full-run saving intervals are positive in both domains. This comparison measures GPU-hours against the evaluated single-refresh schedule. The wall-time comparison against direct verifier-induced allocation uses a different cost endpoint and comparator. Table 38 lists audit-only expenses separately from both training branches. These findings establish the performance and cost tradeoff for the evaluated single-refresh schedule.