MASHAP-LLM: Multi-Agent Fine-Tuning with Context-Aware Shapley-Value Reward Allocation in Large Language Models

Anonymous ACL submission

Abstract

Multi-agent LLM fine-tuning with verifier feedback often yields sparse system-level rewards, making agent-level credit assignment difficult. We propose MASHAP-LLM, a Shapley-based reward-allocation framework that constructs coalition-value games and converts verifier rewards into role-aware agent-level learning signals. Within MASHAP-LLM, LLM-SHAP learns a contextual Shapley-value estimator from verifier-derived counterfactual coalition labels; after the estimator is frozen, it predicts coalition values from mask-conditioned contexts built from the realized grand-coalition trajectory and produces budget-preserving Shapley-shaped rewards for agent-level PPO updates. We analyze budget preservation, estimator-error propagation, and conditional local convergence of the resulting surrogate objective. With 3B-scale Qwen2.5-family backbones, MASHAP-LLM achieves average gains of 2.51 and 4.36 points over the strongest non-MASHAP-LLM baseline under the dual-agent and triple-agent settings, respectively; its best scores reach 54.80% on MATH500, 86.12% on CMATH, 82.49% on GSM8K, and 6.47% on Codeforces.

1 Introduction

Large language models (LLMs) have achieved strong performance on verifiable reasoning tasks such as mathematics and code generation, especially when optimized with reinforcement learning from correctness or unit-test feedback (Shao et al., 2024; Guo et al., 2025; Le et al., 2022). A natural extension is to organize multiple LLM agents into sequential role-specialized collaboration, where different agents analyze, solve, and verify the same task instance (Li et al., 2023; Wu et al., 2024; Liao et al., 2025). Yet verifier-based multi-agent LLM fine-tuning exposes a credit-assignment challenge: the environment usually returns only one terminal system-level reward, which indicates whether the complete multi-agent trajectory succeeds but not how the reward should be allocated to individual agents. This motivates Shapley-based reward allocation, where the sparse system-level reward is decomposed into agent-level learning signals according to each agent's average marginal contribution over coalitions.

Credit-assignment gap. Existing multi-agent LLM fine-tuning methods commonly assign the same system-level reward to every agent (Liao et al., 2025; Park et al., 2025). We denote this equal-allocation baseline as $r_i = R/n$. Equal allocation is simple but mismatched to heterogeneous agent roles: when the Actor carries most answer-producing responsibility, identical credit under-rewards high-contribution agents and over-rewards low-contribution ones, leading to biased advantage estimates and unstable PPO updates (Schulman et al., 2017). Figure 1 illustrates this gap.

Recent credit-assignment methods for multi-agent LLM reinforcement fine-tuning, such as C3 (Chen et al., 2026), CCPO (Li et al., 2026b), and SHARP marginal attribution (Li et al., 2026a), improve over equal allocation by introducing local counterfactual or masked-marginal credit signals. However, these signals remain local rather than full-coalition allocations: C3 uses a contextual decision-level baseline, CCPO estimates the one-agent-removal marginal $v(\mathcal{N}) - v(\mathcal{N} \setminus \{i\})$, and SHARP measures the score drop after ablating one agent from a realized trajectory. None of these constructs a coalition-value function $v : 2^{\mathcal{N}} \to \mathbb{R}$ over all coalitions and then applies the Shapley operator to average marginal contributions across the full coalition set.

Approach and contributions. To address the credit-assignment gap under sparse system-level verifier rewards, we propose MASHAP-LLM, a Shapley-based reward-allocation framework for multi-agent LLM reinforcement fine-tuning. The

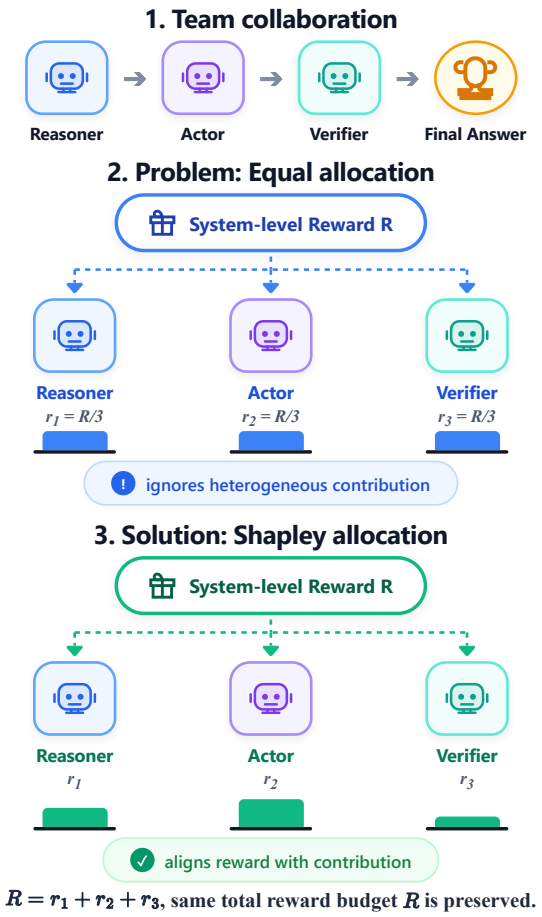

Figure 1: Equal versus Shapley reward allocation. Equal allocation assigns identical credit to all agents, while Shapley allocation redistributes the same total reward according to average marginal contribution.

central idea is to make marginal contribution operational for LLM agents: MASHAP-LLM first acquires a coalition-value game that measures the contribution of different agent subsets, and then applies Shapley-based allocation to convert a terminal system-level reward into role-aware agent-level learning signals. This provides a full-coalition alternative to equal allocation, one-agent-removal credit, and realized-trajectory masking, which provide useful local signals but do not construct the full coalition-value game used by Shapley allocation.

First, we propose MASHAP-LLM, a Shapley-based reward allocation framework for multi-agent LLM reinforcement fine-tuning. MASHAP-LLM operationalizes the marginal-contribution principle by separating credit assignment into coalition-value acquisition and Shapley-based reward allocation, converting a terminal system-level reward into role-aware agent-level learning signals. **Second**, we introduce LLM-SHAP, a learned contextual Shapley-value estimator within MASHAP-LLM. LLM-SHAP predicts coalition values from tokenized coalition-conditioned rollout contexts and converts the induced coalition-value game into agent-level Shapley rewards. This design allows the Shapley allocation layer to use context-informed coalition values under sparse verifier feedback, rather than relying only on hard verifier outcomes. **Third**, we introduce a new formalization for verifier-based multi-agent LLM fine-tuning that is tailored to reward allocation, coupling a cooperative text-action Markov game with an induced per-step coalition game for Shapley allocation. This formulation supports our analysis of budget preservation, estimator-error propagation, and conditional local convergence, and experiments show positive mean gains over equal allocation and controlled local credit-assignment baselines under matched experimental settings.

2 Problem Formulation

2.1 Cooperative Multi-Agent LLM Fine-Tuning under System-Level Rewards.

We study cooperative multi-agent LLM fine-tuning in which a team of heterogeneous LLM agents collaborates to solve a task and receives a system-level reward for the resulting multi-agent trajectory. The system-level reward evaluates the collaborative outcome as a whole, but it does not specify how much each individual agent contributed. The central problem is therefore agent-level credit assignment: how to convert a system-level reward into role-aware learning signals for individual agents.

We introduce a new model of collaborative multi-agent LLM fine-tuning as a cooperative text-action Markov game (Oliehoek et al., 2016)

$$\mathcal{E} = \langle \mathcal{V}, \mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{G} \rangle. \quad (1)$$

where $\mathcal{V}$ is the LLM token vocabulary; $\mathcal{N} = \{1, \ldots, n\}$ is the set of heterogeneous agents; $\mathcal{S}$ is the textual interaction-state space; $\mathcal{A} = \prod_i \mathcal{A}_i$, with $a_i \in \mathcal{A}_i$ denoting an agent-level text action sampled from π_{θ_i}; $\mathcal{T}$ is the state-transition function; $\mathcal{R} : \mathcal{S}_{\text{terminal}} \to [0, R_{\max}]$ is the system-level reward function; and $\mathcal{G}$ specifies the collaboration protocol. The protocol $\mathcal{G}$ can specify different multi-agent interaction structures. In this paper, we use a fixed sequential role-based protocol, so agents act in a causal order and the transition

reduces to deterministic transcript concatenation, $s_i = s_{i-1} \| a_i$. In the verifiable reasoning and code-generation settings studied here, the system-level reward is returned by an external verifier at the terminal state.

State and decision unit. The task state purely records the environment interaction up to time t, $s_t = (q, \text{Actions}_{<t}, \text{Judge}_{<t})$, with q the user prompt; it is independent of training, gradient, or optimizer information. Each agent action $a_{t,i}$ is generated autoregressively as a sequence of tokens, but tokens are only the representational substrate of LLM actions: the RL decision unit in MASHAP-LLM is the agent-level text action $a_{t,i}$, not an individual token, and Shapley credit is assigned to the complete agent action.

Verifier reward and induced coalition game. Let $\text{Verifier}(q, \boldsymbol{a}) \in [0, R_{\max}]$ denote the deterministic score returned by the environment verifier for the original question q and a joint action trajectory $\boldsymbol{a}$. For a coalition $C \subseteq \mathcal{N}$ and a fixed policy snapshot $\boldsymbol{\pi_\theta}$, we define an intervention coalition value through coalition-conditioned rollout:

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

where agents in C participate in the rollout and agents outside C are replaced by a fixed absence message. Thus, the cooperative text-action Markov game describes the sequential multi-agent interaction, while each verifier-scored rollout step induces a static cooperative coalition game $v_{q,\boldsymbol{a}} : 2^{\mathcal{N}} \to \mathbb{R}$ used for Shapley reward allocation. The verifier-induced reference value is denoted by

$$v^{\text{env}}_{q,\boldsymbol{a}}(C) = \text{Verifier}(q, \boldsymbol{a}^C). \tag{3}$$

A learned allocation module may instead use an estimator $\hat{v}_\psi(x_C)$ to define a learned coalition-value table. In LLM-SHAP, the supervised training contexts may contain coalition-specific counterfactual rollouts, whereas frozen allocation uses mask-conditioned textual contexts built from the realized grand-coalition trajectory; the proposed instantiation is described in Section 3.1.

Shapley allocation. Given a coalition-value function $v_{q,\boldsymbol{a}} : 2^{\mathcal{N}} \to \mathbb{R}$, the Shapley reward (Shapley, 1953) assigned to agent i is

$$\begin{aligned} \phi_i(v_{q,\boldsymbol{a}}) &= \sum_{C \subseteq \mathcal{N} \setminus \{i\}} w_{|C|} \Delta_i(C), \\ \Delta_i(C) &= v_{q,\boldsymbol{a}}(C \cup \{i\}) - v_{q,\boldsymbol{a}}(C), \end{aligned} \tag{4}$$

where

$$w_k = \frac{k!(n-k-1)!}{n!}.$$

Thus, Shapley allocation assigns each agent its average marginal contribution over all predecessor coalitions.

2.2 System Objective and Shapley-Shaped Local Surrogates

The original system objective is the expected verifier return,

$$J_{\text{sys}}(\boldsymbol{\pi}) = \mathbb{E}_{p_{\boldsymbol{\pi}}}\Big[\textstyle\sum_{t=0}^{T-1} \text{Verifier}(q_t, \boldsymbol{a}_t)\Big]. \tag{5}$$

At each verifier-scored step, MASHAP-LLM converts the system-level score into per-agent Shapley-shaped rewards

$$r^{\text{Shapley}}_{t,i} = \phi_i(v_{q_t,\boldsymbol{a}_t}),$$

and updates each agent with a local surrogate

$$J_{\text{Sh},i}(\theta_i; \theta_{-i}) = \mathbb{E}_{p_{\boldsymbol{\pi}}}\Big[\textstyle\sum_{t=0}^{T-1} r^{\text{Shapley}}_{t,i}\Big]. \tag{6}$$

We write the stacked local learning signal as

$$\nabla J_{\text{Sh}} = (\nabla_{\theta_i} J_{\text{Sh},i})_{i=1}^n.$$

The theoretical analysis in Section 4 studies budget preservation, estimator-error propagation, and conditional convergence of this Shapley-shaped local surrogate system.

3 The MASHAP-LLM Framework and Algorithm

MASHAP-LLM places Shapley-based reward allocation between system-level verification and agent-level PPO optimization. Given a realized multi-agent trajectory with verifier score R_t, the framework constructs or predicts a coalition-value table, applies Shapley allocation, rescales the resulting credits into agent-level rewards, and uses them for one-step PPO updates.

Algorithm 1 gives the complete MASHAP-LLM fine-tuning algorithm with LLM-SHAP-based coalition-value estimation and Shapley-shaped PPO updates. The verifier-induced and static-masked variants are used only for ablation and are described in Appendix D.4. Here, CoalTrain and CoalInfer denote coalition-context construction before and after estimator freezing, respectively; InferV, FitV, and Alloc denote learned value inference, estimator fitting, and Shapley-based reward allocation. We

Algorithm 1 MASHAP-LLM fine-tuning with LLM-SHAP allocation.

Input: Agents $\mathcal{N}$, policies $\{\pi_{\theta_i}\}_{i=1}^{n}$, verifier Verifier, estimator $\hat{v}_\psi$, critic V_ω, rollout horizon T, PPO hyperparameters, normalization floor τ_{floor}, freezing rule λ_{stop}
Output: Updated policies $\{\pi_{\theta_i}\}_{i=1}^{n}$, estimator $\hat{v}_\psi$, critic V_ω

Initialize replay buffer $\mathcal{B} \leftarrow \emptyset$ and frozen $\leftarrow$ False
for episode $= 0, 1, \ldots$ **do**
  $s_0 \leftarrow \text{Env.reset}()$
  for $t = 0, \ldots, T-1$ **do**
    for $i = 1, \ldots, n$ **do**
      Form agent input context $c_{t,i}$ from q_t and previous actions $a_{t,1:i-1}$
      Sample text action $a_{t,i} \sim \pi_{\theta_i}(\cdot \mid c_{t,i})$
      $\ell_{t,i} \leftarrow \log \pi_{\theta_i}(a_{t,i} \mid c_{t,i})$
      $V_{t,i} \leftarrow V_\omega(c_{t,i})$
    $\boldsymbol{a}_t^{\mathcal{N}} \leftarrow (a_{t,1}, \ldots, a_{t,n})$
    $R_t \leftarrow \text{Verifier}(q_t, \boldsymbol{a}_t^{\mathcal{N}})$
    if not frozen **then**
      $\mathcal{D}_t^{\text{coal}} \leftarrow \textsc{CoalTrain}(q_t, \boldsymbol{a}_t^{\mathcal{N}}, R_t, \text{Verifier})$
    else
      $\mathcal{D}_t^{\text{coal}} \leftarrow \textsc{CoalInfer}(q_t, \boldsymbol{a}_t^{\mathcal{N}})$
    $\widehat{\mathcal{V}}_t \leftarrow \textsc{InferV}(\mathcal{D}_t^{\text{coal}}, \hat{v}_\psi)$
    $\boldsymbol{r}_t \leftarrow \textsc{Alloc}(\widehat{\mathcal{V}}_t, R_t, \tau_{\text{floor}})$
    if not frozen **then**
      $\psi \leftarrow \textsc{FitV}(\psi, \mathcal{D}_t^{\text{coal}})$
      frozen $\leftarrow \textsc{FreezeCheck}(\hat{v}_\psi; \lambda_{\text{stop}})$
    Store $\{c_{t,i}, a_{t,i}, r_{t,i}, V_{t,i}, \ell_{t,i}\}_{i=1}^{n}$ in $\mathcal{B}$
    $s_{t+1} \leftarrow \text{Env.transition}(s_t, \boldsymbol{a}_t^{\mathcal{N}})$
  Compute $A_{t,i} = r_{t,i} - V_{t,i}$ for samples in $\mathcal{B}$
  Update $\{\pi_{\theta_i}\}_{i=1}^{n}$ and V_ω by the PPO-Clip objective in Eq. (9)
  Clear replay buffer $\mathcal{B}$

use environment-level counterfactual rollouts only for early estimator supervision and ablation; after freezing, LLM-SHAP uses masked contextual prediction from the realized grand-coalition trajectory without subcoalition rollouts or verifier calls. Their full procedures are provided in Appendix D.

Estimator freezing. LLM-SHAP is updated only during early estimator fitting and is frozen once the stopping rule λ_{stop} is met. After freezing, allocation uses masked contexts from the realized grand-coalition trajectory and stops all strict-subcoalition rollouts and verifier calls. The detailed freezing rule is provided in Appendix E.7.1.

3.1 LLM-SHAP: A Learned Contextual Coalition-Value Estimator

LLM-SHAP is the learned allocation-time instantiation of MASHAP-LLM. At the module level, LLM-SHAP outputs agent-level Shapley-shaped rewards; internally, it first predicts coalition values and then applies the Shapley-value allocator. For each coalition $C \subseteq \mathcal{N}$, the estimator predicts a coalition value $\hat{v}_\psi(C)$, and the allocator maps the complete predicted coalition-value table to an agent-level reward vector.

During early estimator fitting, each coalition C is paired with a coalition-conditioned text context x_C and a verifier-derived target

$$y_C = v^{\text{env}}_{q_t, \boldsymbol{a}_t}(C) = \text{Verifier}(q_t, \boldsymbol{a}_t^C),$$

with the grand-coalition target cached as $R_t = \text{Verifier}(q_t, \boldsymbol{a}_t^{\mathcal{N}})$. These verifier-derived labels train the estimator $\hat{v}_\psi(x_C)$ to approximate contextual coalition values. Verifier-induced coalition values are used only for early LLM-SHAP supervision and non-learned ablation references; they are not the frozen allocation-time mechanism in the reported MASHAP-LLM results.

After the estimator is frozen, strict-subcoalition rollouts and verifier calls are stopped. For each coalition mask m_C, LLM-SHAP constructs a masked textual context from the single realized grand-coalition trajectory,

$$x^{\text{infer}}_{t,C} = \text{MaskCtx}(q_t, \boldsymbol{a}_t^{\mathcal{N}}, m_C),$$

and predicts

$$\hat{v}_\psi(C) = \text{LLMShap}_\psi(x^{\text{infer}}_{t,C}).$$

The resulting predicted table $\{\hat{v}_\psi(C) : C \subseteq \mathcal{N}\}$ is then converted into agent-level rewards through Shapley allocation and reward rescaling. Implementation details of the estimator backbone, value head, freezing rule, and coalition-context construction are provided in Appendix D, Appendix E.7.1, and Appendix E.7.

3.2 Shapley Reward Normalization

Given the predicted coalition-value table, we apply Eq. (4) to obtain the raw Shapley credits $\hat{\boldsymbol{\phi}}$. To preserve the realized verifier reward budget, we rescale these credits by the verifier score R_t:

$$r^{\text{LLM-SHAP}}_{t,i} = \begin{cases} \dfrac{R_t \hat{\phi}_i}{\hat{\Sigma}}, & \text{if } |\hat{\Sigma}| \geq \tau_{\text{floor}} \\ & \text{and } \text{sign}(\hat{\Sigma}) = \text{sign}(R_t), \\ R_t/n, & \text{otherwise.} \end{cases} \tag{7}$$

where $\hat{\Sigma} = \sum_j \hat{\phi}_j$. The fallback branch is a numerical safeguard for unstable normalization and is not treated as a separate modeling component.

3.3 One-Step Action-Level Advantage and PPO-Clip Update

Sequential execution induces an augmented timeline indexed by (t, i), where t denotes the environment step and i denotes the acting agent in the

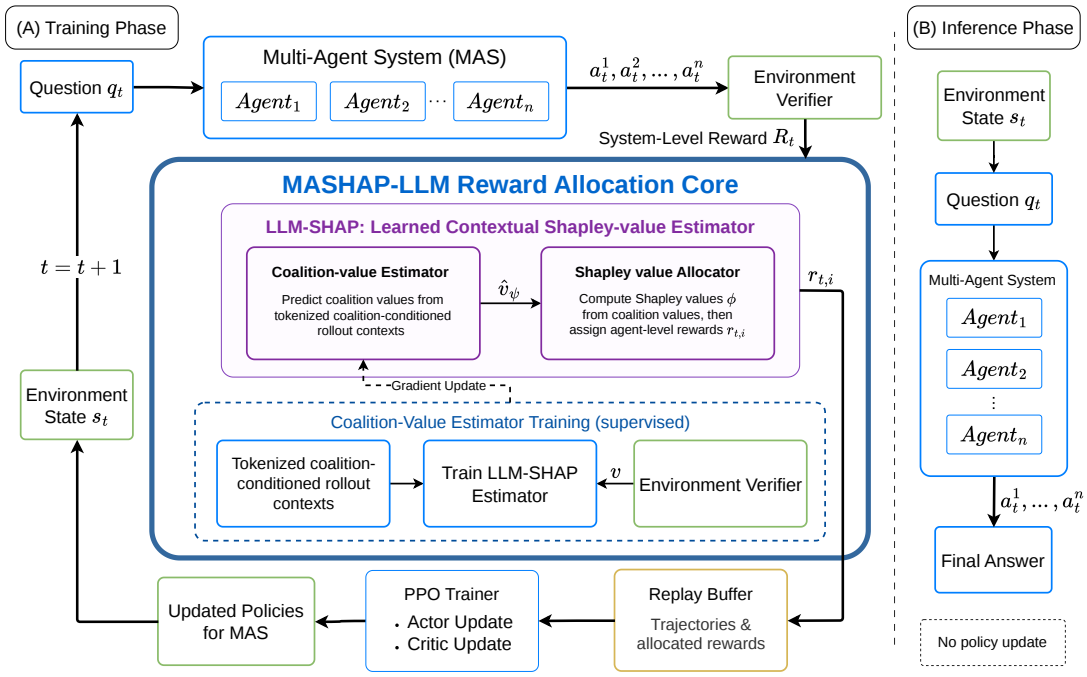

Figure 2: MASHAP-LLM framework with the LLM-SHAP instantiation. MASHAP-LLM acquires contextual coalition values and applies Shapley-based reward allocation to transform system-level verifier feedback into role-aware agent-level learning signals.

sequential role order. The value critic uses the agent input context $c_{t,i}$ as the state-dependent baseline input before agent i generates its text action. Since each rollout step produces a complete realized multi-agent trajectory and an immediate allocation target derived from the realized verifier reward and the coalition-value allocator, MASHAP-LLM uses a one-step action-level advantage estimator rather than a temporal return estimator:

$$A_{t,i} = r_{t,i}^{\text{Shapley}} - V_\omega(c_{t,i}). \tag{8}$$

Here $V_\omega(c_{t,i})$ serves as a state-dependent baseline for variance reduction, and the critic is trained to regress directly to the allocated Shapley reward $r_{t,i}^{\text{Shapley}}$. This design keeps the policy-update signal aligned with the per-step coalition-value game: the Shapley-value allocator already converts the realized verifier score into agent-level rewards according to marginal coalition contributions, so no temporal bootstrapping is introduced.

The per-agent PPO-Clip objective is (Schulman et al., 2017)

$$\mathcal{L}_i^{\text{PPO}} = \mathbb{E}\Big[\min\Big(\rho_{t,i}A_{t,i}, \text{clip}_{\varepsilon_{\text{clip}}}(\rho_{t,i})A_{t,i}\Big)\Big] - c_v\Big(V_\omega(c_{t,i}) - r_{t,i}^{\text{Shapley}}\Big)^2, \tag{9}$$

where

$$\rho_{t,i} = \frac{\pi_{\theta_i}(a_{t,i} \mid c_{t,i})}{\pi_{\theta_i,\text{old}}(a_{t,i} \mid c_{t,i})}$$

and $\varepsilon_{\text{clip}} = 0.2$.

4 Theoretical Analysis

We present three theoretical properties of MASHAP-LLM. The first establishes that Shapley-based allocation preserves the realized verifier reward budget. The second characterizes how coalition-value estimation error propagates to the normalized LLM-SHAP rewards. The third gives a conditional PPO convergence bound for the Shapley-shaped local surrogate under the one-step contextual-bandit training regime. All assumptions and full proofs are provided in Appendix B and Appendix C.

Theorem 1 (Budget-preserving Shapley allocation)**.** *Let $R_t := \text{Verifier}(q_t, \boldsymbol{a}_t^{\mathcal{N}})$ denote the verifier score of the realized grand-coalition trajectory at step t. For direct verifier-induced Shapley allocation, the allocated rewards satisfy*

$$\sum_i r_{t,i}^{\text{Shapley}} = R_t = \text{Verifier}(q_t, \boldsymbol{a}_t^{\mathcal{N}}).$$

For LLM-SHAP*, whenever the non-degenerate rescaling rule in Eq. (7) is applied, the normalized*

rewards also satisfy

$$\sum_i r_{t,i}^{\text{LLM-SHAP}} = R_t.$$

If the implementation safeguard is triggered, the fallback rule $r_{t,i} = R_t/n$ also preserves the same reward budget.

The proof follows from Shapley efficiency and the budget-preserving normalization rule; see Appendix C.

Theorem 2 (Estimator-error propagation under non-degenerate rescaling)**.** *Let ϕ^* denote the exact Shapley allocation induced by the verifier-defined coalition game. Under the bounded estimator-error and non-degenerate rescaling conditions in Appendix B, the post-normalized* LLM-SHAP *reward vector $\boldsymbol{r}$ satisfies*

$$|r_i - \phi_i^*| \leq (2 + K_R R_{\max})\varepsilon_{\text{est}} + 2K_R\varepsilon_{\text{est}}^2,$$
$$K_R = \frac{2n}{\tau_{\text{floor}}}.$$

Before normalization, the unnormalized Shapley estimate satisfies

$$|\hat{\phi}_i - \phi_i^*| \leq 2\varepsilon_{\text{est}}.$$

If the numerical safeguard $r_{t,i} = R_t/n$ is triggered, only budget preservation is claimed.

This result shows that learned coalition-value estimation introduces a controlled distortion into the allocated rewards when the estimator error and normalization denominator are controlled; see Appendix C.4 for the full proof.

Theorem 3 (Conditional convergence of the Shapley-shaped local surrogate)**.** *Under the one-step contextual-bandit training regime and the PPO assumptions in Appendix B, let*

$$\Delta_{\text{est}} = \begin{cases} 0, & \textit{verifier-induced Shapley}, \\ K_R^2\varepsilon_{\text{est}}^2, & \text{LLM-SHAP}. \end{cases}$$

Then, with $\eta = K^{-1/2}$ over K optimization updates,

$$\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\left[\|\nabla J_{\text{Sh}}(\theta_k)\|^2\right] \leq \mathcal{O}(K^{-1/2}) + \mathcal{O}(\varepsilon_{\text{clip}}^2) + \mathcal{O}(\delta_{\text{clip}}) + \mathcal{O}(\Delta_{\text{est}}).$$

For LLM-SHAP*, this bound applies to non-degenerate rescaling steps. Safeguard steps preserve budget balance but are not covered by the closeness-based estimator-error term. The result is a convergence statement for the Shapley-shaped local surrogate system, not for the original system objective J_{sys}.*

The proof follows the standard smooth non-convex stochastic-approximation argument for PPO clipped surrogates (Liu et al., 2019; Bhandari and Russo, 2024), with the LLM-SHAP estimator error entering through Theorem 2; see Appendix C.5.

Interpretation. Together, Theorems 1–3 show that MASHAP-LLM preserves the realized verifier-reward budget, bounds the distortion introduced by learned coalition-value estimation under non-degenerate normalization, and yields a conditional stationarity bound for the one-step Shapley-shaped local surrogate. These results justify the reward-allocation mechanism under the stated assumptions, but they do not imply global optimality of the original system objective or guarantee superior final task accuracy.

5 Experiments

5.1 Experimental Setup

Tasks and benchmarks. We evaluate MASHAP-LLM on mathematical reasoning and competitive programming. For mathematical reasoning, models are trained on MATH (Hendrycks et al., 2021) or CMATH (Wei et al., 2023) and evaluated on MATH500, CMATH, and GSM8K (Cobbe et al., 2021), covering in-domain, cross-lingual, and out-of-domain arithmetic generalization. For competitive programming, models are trained on Codeforces and evaluated on a held-out Codeforces split.

Backbones and adaptation. We use Qwen2.5-Coder-3B-Instruct (Hui et al., 2024) for mathematical reasoning and Qwen2.5-3B-Instruct (Yang et al., 2025) for Codeforces. All multi-agent systems use role-specific LoRA (Hu et al., 2022) adapters with the same base model family within each task setting. Full hyperparameters, agent prompts, data splits, and hardware details are provided in Appendix E.

Baselines and metrics. We compare against the zero-shot SFT backbone, Equal-allocation MARFT-A (Liao et al., 2025), and controlled local credit-assignment baselines: C3 (Chen et al., 2026), CCPO (Li et al., 2026b), and SHARP (Li

SFT Baselines (No MARL)		Zero-shot Capability			
		MATH500	**CMATH**	**GSM8K**	**Codeforces**
SFT Backbone (Direct Inference, $N = 2$)		44.80	79.97	74.77	4.33
SFT Backbone (Direct Inference, $N = 3$)		40.40	79.87	75.97	1.83
Framework	**Credit Assignment**	**Trained on MATH**			**Trained on CMATH**			**Trained on Codeforces**
		MATH500 *(ID)*	**CMATH** *(OOD)*	**GSM8K** *(OOD)*	**MATH500** *(OOD)*	**CMATH** *(ID)*	**GSM8K** *(OOD)*	**Codeforces** *(ID)*
Dual-Agent Configuration *($N = 2$)*								
MARFT-A	Equal (Global Reward)	46.10 ± 1.16	80.26 ± 0.88	76.14 ± 0.46	46.80 ± 0.61	80.23 ± 0.96	74.97 ± 0.35	4.52 ± 0.25
SHARP	Static masked marginal credit	46.50 ± 0.52	80.34 ± 1.03	76.42 ± 0.76	46.50 ± 0.31	80.12 ± 0.62	74.61 ± 0.82	4.51 ± 0.31
CCPO	Leave-One-Out (LOO)	47.60 ± 0.58	82.40 ± 1.02	78.35 ± 0.88	48.90 ± 0.40	82.37 ± 0.71	77.72 ± 0.78	4.60 ± 0.33
C3	Decision-level CF	48.60 ± 0.73	82.28 ± 1.05	78.51 ± 0.88	49.40 ± 0.30	82.44 ± 0.57	77.60 ± 0.93	4.74 ± 0.41
MASHAP-LLM	**LLM-SHAP Allocation (Ours)**	$\mathbf{52.30 \pm 0.43}$	$\mathbf{84.79 \pm 0.87}$	$\mathbf{80.61 \pm 0.73}$	$\mathbf{52.60 \pm 0.60}$	$\mathbf{84.15 \pm 0.72}$	$\mathbf{81.15 \pm 0.91}$	$\mathbf{5.79 \pm 0.24}$
Triple-Agent Configuration *($N = 3$)*								
MARFT-A	Equal (Global Reward)	46.00 ± 0.89	80.72 ± 1.21	77.61 ± 0.79	46.83 ± 0.70	81.37 ± 1.35	76.50 ± 0.53	4.46 ± 0.34
SHARP	Static masked marginal credit	46.10 ± 1.12	80.14 ± 1.23	77.15 ± 0.60	46.52 ± 0.40	81.16 ± 1.14	76.05 ± 0.24	4.41 ± 0.23
CCPO	Leave-One-Out (LOO)	47.70 ± 0.64	80.50 ± 1.32	77.57 ± 1.52	45.90 ± 0.20	82.44 ± 0.84	77.84 ± 0.75	4.74 ± 0.28
C3	Decision-level CF	48.15 ± 1.21	81.02 ± 1.13	78.77 ± 0.80	47.61 ± 0.60	83.45 ± 1.10	78.05 ± 0.56	4.83 ± 0.22
MASHAP-LLM	**LLM-SHAP Allocation (Ours)**	$\mathbf{54.47 \pm 0.85}$	$\mathbf{85.63 \pm 1.02}$	$\mathbf{82.45 \pm 0.78}$	$\mathbf{54.80 \pm 0.40}$	$\mathbf{86.12 \pm 1.31}$	$\mathbf{82.49 \pm 0.83}$	$\mathbf{6.47 \pm 0.37}$

Table 1: Performance on mathematical reasoning and Codeforces benchmarks. Math scores are Pass@1 accuracy; Codeforces scores are average unit-test pass rate. Fine-tuned methods report mean $\pm$ standard deviation over five seeds. The upper block shows zero-shot SFT backbones; the lower block compares credit-assignment methods. Best results are bolded.

et al., 2026a). All fine-tuning baselines share the same verifier-based multi-agent scaffold, backbone, prompts, LoRA configuration, optimizer, rollout budget, checkpoint-selection rule, and evaluation protocol; they differ only in how the system-level verifier reward is converted into agent-level training rewards. For mathematical reasoning, we report greedy-decoding Pass@1 accuracy. For Codeforces, we report average unit-test pass rate. Implementation details and fairness controls are provided in Appendix E.1.

5.2 Main Results

Table 1 reports the final evaluation results. Across settings trained on MATH, CMATH, and Codeforces, MASHAP-LLM achieves average gains of 2.51 and 4.36 points over the strongest non-MASHAP-LLM baseline under the dual-agent and triple-agent settings, respectively. The best scores reach 54.80% on MATH500, 86.12% on CMATH, 82.49% on GSM8K, and 6.47% on Codeforces.

Figure 3 summarizes relative improvement over the corresponding SFT backbone and shows positive gains across all reported task families and agent-count settings. These results are consistent with the credit-assignment motivation of MASHAP-LLM: learned coalition-value prediction provides more differentiated agent-level rewards than uniform sharing or local marginal-credit signals, especially in heterogeneous multi-agent settings. Because all baselines use matched training and evaluation settings, the observed differences primarily reflect the credit-construction mechanism rather than changes in model scale, prompting, optimizer, or evaluation protocol. Paired seed-level reliability analysis and additional LLM-SHAP diagnostics are provided in Appendix K and Appendix J.

5.3 Ablation Study

Table 2 isolates the two design choices in LLM-SHAP under both dual-agent and triple-agent settings. All variants keep the same Shapley allocation layer and differ only in how the coalition-value table is obtained: learned contextual prediction with early counterfactual-rollout supervision, direct verifier-induced coalition values, or static masked coalition values. The results show a consistent ordering across both agent configurations. The full MASHAP-LLM / LLM-SHAP variant performs best, indicating that learned contextual coalition-value prediction improves the allocation signal beyond directly using verifier-scored coalition values. Verifier-induced Shapley also outperforms the static-masked variant, showing that coalition-specific counterfactual rollout provides stronger supervision or reference values than post-hoc mask-

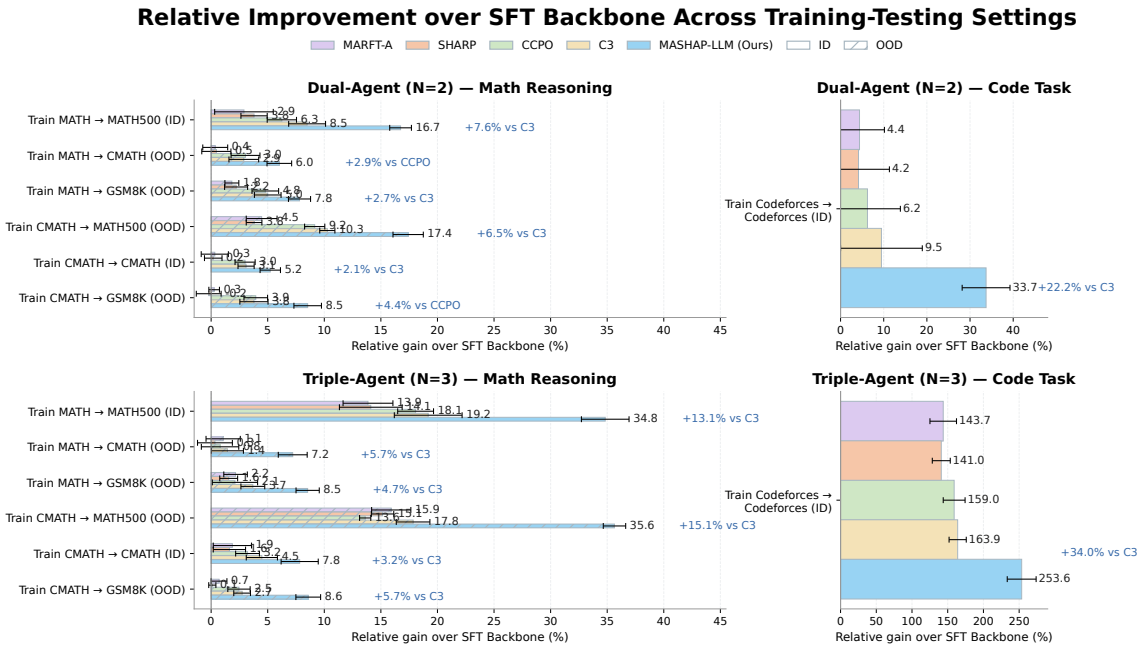

Figure 3: Relative improvement of RL fine-tuning frameworks over the corresponding SFT backbone. Gains are computed as $(\text{Method} - \text{Backbone})/\text{Backbone} \times 100\%$.

Variant	MATH500	CMATH	GSM8K	Code
Dual-Agent Configuration ($N = 2$)				
Ours (L+CF-train)	**52.30±0.43**	**84.79±0.87**	**80.61±0.73**	**5.79±0.24**
Verifier (CF)	48.39±0.16	81.25±1.03	77.33±0.31	4.64±0.53
Static (No L/CF)	47.26±0.47	75.42±1.55	71.90±0.11	3.84±0.46
Triple-Agent Configuration ($N = 3$)				
Ours (L+CF-train)	**54.47±0.85**	**85.63±1.02**	**82.45±0.78**	**6.47±0.37**
Verifier (CF)	49.60±0.94	82.79±1.05	79.45±0.62	5.47±0.16
Static (No L/CF)	47.20±0.70	77.20±1.20	73.00±1.00	4.76±0.40

Table 2: Ablation study under $N = 2$ and $N = 3$ settings. All variants use Shapley allocation; L and CF-train denote learned value estimation and counterfactual-rollout supervision before freezing.

ing of the realized trajectory. Overall, the ablation supports the proposed design: counterfactual-rollout-derived supervision before freezing, followed by frozen masked-context coalition-value prediction for allocation. Additional internal ablations of LLM-SHAP are provided in Appendix J.2.

6 Conclusion

This paper studies agent-level reward allocation in verifier-based multi-agent LLM reinforcement fine-tuning. We introduced MASHAP-LLM, a Shapley-based reward-allocation framework that converts sparse system-level verifier rewards into role-aware agent-level learning signals. Within this framework, LLM-SHAP learns coalition values from tokenized coalition-conditioned counterfactual rollout contexts during early estimator fitting and then produces Shapley-shaped rewards through frozen mask-conditioned coalition-value prediction and reward rescaling. Theoretically, MASHAP-LLM preserves the verifier reward budget, bounds the effect of learned coalition-value estimation error under non-degenerate rescaling, and admits a conditional PPO convergence analysis for the Shapley-shaped local surrogate. Empirically, across settings trained on MATH, CMATH, and Codeforces, MASHAP-LLM achieves positive mean gains over equal allocation and controlled local credit-assignment baselines. Future work will explore approximate Shapley estimation for larger agent teams, long-horizon multi-agent RL, and more adaptive coalition-value estimation mechanisms.

7 Limitations

Empirical regime. Our experiments focus on verifier-based mathematical reasoning and competitive programming under a short-horizon, one-step contextual-bandit training regime. The current results therefore do not establish performance in open-ended dialogue, tool-use agents, cross-backbone heterogeneous teams, or long-horizon multi-agent RL with temporally bootstrapped per-agent value functions.

Coalition-value supervision and estimator risk. LLM-SHAP is trained from verifier-derived coalition supervision constructed through coalition-conditioned counterfactual rollout during the early fitting phase. This design depends on the quality of the verifier-induced coalition labels and may be affected by absence-message interventions, distribution shift, value-head instability, and estimator error ε_{est}. When the normalization denominator is unreliable, the method falls back to budget-preserving equal allocation, which preserves reward mass but does not provide Shapley-specific credit information.

Theory scope. The main theoretical results concern budget preservation, estimator-error propagation, and conditional convergence of the Shapley-shaped local surrogate. They do not imply global optimality of the original system objective or guarantee superior final task accuracy. The auxiliary Core result in Appendix C.9 is conditional on convex verifier-induced coalition games, and the LLM-SHAP estimator-error guarantee applies only under the estimator-error and non-degenerate rescaling conditions in Assumption A8.

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. AI assistants were used only for language polishing, grammar checking.

References

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

Jalaj Bhandari and Daniel Russo. 2024. Global optimality guarantees for policy gradient methods. *Operations Research*, 72(5):1906–1927.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

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

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

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. 2022. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 21314–21328.

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

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

Yanming Li, Xuelin Zhang, WenJie Lu, Ziye Tang, Maodong Wu, Haotian Luo, Tongtong Wu, Zijie Peng, Hongze Mi, Yibo Feng, and 1 others. 2026a. Who deserves the reward? sharp: Shapley credit-based optimization for multi-agent system. *arXiv preprint arXiv:2602.08335*.

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

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

Boyi Liu, Qi Cai, Zhuoran Yang, and Zhaoran Wang. 2019. Neural proximal/trust region policy optimization attains globally optimal policy. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA. Curran Associates Inc.

Frans A Oliehoek, Christopher Amato, and 1 others. 2016. *A concise introduction to decentralized POMDPs*, volume 1. Springer.

Chanwoo Park, Seungju Han, Xingzhi Guo, Asuman E. Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. 2025. 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)*, pages 30215–30248, Vienna, Austria. Association for Computational Linguistics.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

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

Lloyd S Shapley. 1971. Cores of convex games. *International journal of game theory*, 1(1):11–26.

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

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

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. 2024. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, and 1 others. 2025. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

A Related Work

Multi-agent LLM reinforcement fine-tuning. Recent work has begun to extend reinforcement fine-tuning from single LLMs to multi-agent LLM systems. MARFT-A (Liao et al., 2025) establishes a multi-agent fine-tuning setup with action-level and token-level optimization, while MAPoRL (Park et al., 2025) studies post-co-training for collaborative LLMs. These methods provide important training scaffolds, but their reward path remains system-level or equally shared, leaving agent-level credit assignment under-specified. Single-agent reasoning RL methods such as GRPO optimization and DeepSeek reasoning training (Shao et al., 2024; Guo et al., 2025) further motivate outcome-level verifier rewards, but do not solve credit allocation among heterogeneous agents.

Local and counterfactual credit assignment. Credit assignment has long been studied in MARL through counterfactual baselines. COMA (Foerster et al., 2018) estimates agent-specific counterfactual advantages for classical MARL, but it is not designed for text-valued LLM actions. Recent multi-agent LLM methods adapt this idea: C3 (Chen et al., 2026) estimates contextual decision-level counterfactual baselines under PPO training, CCPO (Li et al., 2026b) uses one-agent-removal credit $v(\mathcal{N}) - v(\mathcal{N} \setminus \{i\})$ in a GRPO objective, and SHARP marginal attribution (Li et al., 2026a) uses static masking to estimate one-agent marginal effects in realized trajectories. These methods are strong local credit-assignment baselines, but they do not evaluate a full cooperative game $v : 2^{\mathcal{N}} \to \mathbb{R}$ over all coalitions, which is required for full coalition-value-based Shapley allocation. Table 3 summarizes whether each method uses environment-level counterfactual information and whether it performs full-set Shapley allocation. The key distinction of MASHAP-LLM is the combination of verifier-derived coalition supervision before freezing and full-set Shapley allocation over frozen LLM-SHAP predictions after freezing: LLM-SHAP learns contextual coalition values from coalition-conditioned rollout labels and then applies exact Shapley allocation to a predicted coalition-value table built from masked grand-trajectory contexts.

Method	Env-CF	Full-set Shapley	Credit signal
MARFT-A	–	–	Equal allocation of system-level reward
SHARP	–	–	Static masked marginal credit
C3	local	–	Decision-level contextual counterfactual
CCPO	LOO	–	One-agent-removal marginal credit
MASHAP-LLM / LLM-SHAP	train	+	Exact Shapley over frozen masked-context coalition-value predictions

Table 3: Mechanism-level positioning of MASHAP-LLM relative to controlled credit-assignment baselines. **Env-CF** indicates the type of environment-level counterfactual information used, rather than static masking, decision-level local counterfactuals, or one-agent removal. For LLM-SHAP, environment-level counterfactual rollouts are used for early estimator supervision, while frozen allocation uses masked contextual prediction without subcoalition rollouts or verifier calls. **Full-set Shapley** indicates exact enumeration over the full coalition set $2^{\mathcal{N}}$ followed by Shapley allocation.

Shapley value in RL and LLM agents. The Shapley value provides an axiomatic solution for cooperative credit allocation (Shapley, 1953). In classical multi-agent RL, SHAQ (Wang et al., 2022) incorporates Shapley values into Q-learning, and Shapley counterfactual credits (Li et al., 2021) estimate coalition-based credits for discrete-action settings. Recent LLM-focused works, such as SHARP (Li et al., 2026a) and ShapE-GRPO (Ai et al., 2026), also use Shapley-inspired marginal contributions for credit assignment in post-training or candidate-level settings. However, these approaches do not directly address verifier-based multi-agent LLM reinforcement fine-tuning with full coalition enumeration and learned contextual coalition-value shaping. In contrast, our work explicitly constructs a coalition-value game over all agent subsets and uses a learned contextual estimator to produce Shapley-shaped agent rewards during multi-agent fine-tuning. This positions MASHAP-LLM as a full-coalition Shapley allocation method for verifier-based multi-agent LLM fine-tuning.

B Theory Assumptions

This appendix lists the assumptions used in the theoretical analysis. The assumptions are grouped according to their role: the coalition-game definition, the PPO local surrogate analysis, and the learned LLM-SHAP estimator analysis.

A0. Absence-message normalization. For every coalition-value game, we set $v(\emptyset) = 0$. Under direct verifier-induced Shapley allocation, coalitions without an answer-capable agent are assigned value 0 by verifier-protocol convention, since such coalitions cannot produce a final answer for the verifier. This convention makes the coalition-value table well-defined for all $C \subseteq \mathcal{N}$ and fixes how non-participating agents are represented through the absence message. For LLM-SHAP, the same verifier-derived convention may be used to construct supervised targets during early fitting, but the learned estimator is not hard-clamped to zero at frozen inference time and instead predicts from mask-conditioned contexts.

A1. Bounded rewards. For every verifier call,

$$0 \leq \text{Verifier}(q, \boldsymbol{a}) \leq R_{\max}.$$

A2. Smooth policy parameterization. Each policy π_{θ_i} is differentiable, and its score function is L-Lipschitz in θ_i.

A3. Bounded score. For every agent i, text input context c, and text action a,

$$\|\nabla_{\theta_i} \log \pi_{\theta_i}(a \mid c)\| \leq G.$$

A4. One-step optimization regime. The main PPO local convergence result is stated for the empirical training regime in which each allocated Shapley reward is used as an immediate action-level learning signal. The PPO critic is used only as a state-dependent baseline, yielding the one-step advantage

$$A_{t,i} = r_{t,i}^{\text{Shapley}} - V_\omega(c_{t,i}).$$

No discounted temporal return or bootstrapped value-to-go target is used in the main analysis.

A5. Bounded one-step advantage. For the PPO local surrogate analysis, we assume that the one-step advantages used by the PPO critic are bounded:

$$|A_{t,i}| \leq A_{\max}, \quad \forall i, t.$$

This condition is used in the PPO-Clip local smoothness lemma and the local convergence proof.

A6. Bounded local surrogate. For the non-convex stochastic-approximation argument, we assume that the aggregate local surrogate

$$\mathcal{J}_{\mathrm{loc}}(\theta) = \sum_i J_{\mathrm{Sh},i}(\theta_i; \theta_{-i})$$

is upper bounded by a finite constant $J_{\max}$. This condition is used in the summation step of the PPO local convergence proof.

A7. Bounded clipping probability. For each PPO inner update, the probability of leaving the clipping region is bounded:

$$\Pr(|\rho_{t,i} - 1| > \varepsilon_{\mathrm{clip}}) \leq \delta_{\mathrm{clip}}.$$

The effect of off-clip samples is treated as an additional clipping-bias term. This is a standard technical condition for PPO analysis and can be monitored empirically by the fraction of samples whose likelihood ratio leaves the clipping interval.

A8. LLM-SHAP estimator conditions. For the learned LLM-SHAP estimator, we assume the following two conditions on the non-degenerate rescaling steps considered in the bounded-bias and convergence analyses. Here $\hat{v}_\psi(C)$ is shorthand for the frozen estimator's prediction $\hat{v}_\psi(x_C)$ under coalition mask m_C, and $v^{\mathrm{env}}(C)$ is shorthand for the verifier-induced reference value $v^{\mathrm{env}}_{q,\boldsymbol{a}}(C)$ used for supervision or analysis.

Bounded coalition-value estimation error. For every coalition $C \subseteq \mathcal{N}$,

$$|\hat{v}_\psi(C) - v^{\mathrm{env}}(C)| \leq \varepsilon_{\mathrm{est}}.$$

This is a uniform sup-error condition on the LLM-SHAP coalition-value estimator, not on the PPO critic and not on directly predicted Shapley values. In practice, estimator MAE and RMSE can be computed against logged verifier-derived reference labels as empirical diagnostics of prediction quality, although they do not replace the uniform sup-error condition.

Non-degenerate rescaling. For the LLM-SHAP steps covered by the bounded-bias, Core, and convergence statements, the predicted Shapley sum

$$\hat{\Sigma} = \sum_i \hat{\phi}_i$$

satisfies

$$|\hat{\Sigma}| \geq \tau_{\mathrm{floor}} > 0 \quad \text{and} \quad \mathrm{sign}(\hat{\Sigma}) = \mathrm{sign}(R_t).$$

This excludes numerically degenerate or sign-inconsistent normalization steps. When the implementation fallback $r_{t,i} = R_t/n$ is used, we claim budget balance only, not closeness to the verifier-induced Shapley allocation or Core membership.

Relationship between the Markov game and the induced coalition game. The cooperative text-action Markov game describes the sequential multi-agent interaction and policy-optimization process. For each verifier-scored rollout step and fixed policy snapshot, the counterfactual coalition construction further induces a cooperative coalition game

$$v_{q,\boldsymbol{a}} : 2^{\mathcal{N}} \to \mathbb{R},$$

on which Shapley allocation is applied. Thus, the Markov game provides the dynamic training process, while the coalition game provides the static per-step object used for Shapley reward allocation.

C Proofs and Auxiliary Theoretical Results

This appendix provides the full proofs for the three main theoretical results in Section 4, together with auxiliary results used in the local PPO analysis. The main proofs correspond to budget preservation, estimator-error propagation, and conditional convergence of the Shapley-shaped local surrogate. Additional results, including the augmented micro-step view, variance redistribution, clipping diagnostics, and the conditional Core guarantee, are included as auxiliary statements rather than main claims.

Notation follows Section 2. We use s_t for Markov-process and augmented-MDP statements, and use $(q_t, \boldsymbol{a}_t)$ when referring to verifier-induced coalition values, since the verifier is implemented as a deterministic function of the original question and the generated joint action trajectory.

C.1 Preliminaries

For a fixed policy snapshot $\boldsymbol{\pi}_{\boldsymbol{\theta}_k}$ and a realized task instance with question q and joint action trajectory $\boldsymbol{a}$, we write the verifier-induced coalition game as

$$v_{q,\boldsymbol{a}}(C) = \mathrm{Verifier}(q, \boldsymbol{a}^C), \qquad C \subseteq \mathcal{N}.$$

Assumption A0 ensures $v_{q,\boldsymbol{a}}(\emptyset) = 0$. For the grand coalition, the counterfactual trajectory coincides with the realized grand coalition trajectory, so

$$v_{q,\boldsymbol{a}}(\mathcal{N}) = \mathrm{Verifier}(q, \boldsymbol{a}) =: R_t.$$

Although the Markov state is denoted by s_t in the cooperative text-action game formulation, the verifier is called with the original question q_t and the generated joint action $\boldsymbol{a}_t$. Thus, this appendix uses $v_{q,\boldsymbol{a}}$ for verifier-induced coalition games and s_t for Markov-process statements. For frozen LLM-SHAP inference, the verifier-induced game is a reference object for supervision and error analysis; allocation-time coalition values are the predictions obtained from masked grand-trajectory contexts and do not require environment-level subcoalition rollouts.

For LLM-SHAP, the proofs of estimator-error propagation, convergence, and Core slack are restricted to non-degenerate rescaling steps satisfying the LLM-SHAP estimator conditions in Assumption A8. Numerically degenerate steps may still be handled by the implementation safeguard $r_{t,i} = R_t/n$, but those steps are used only to preserve budget balance and are excluded from closeness and Core claims.

C.2 Auxiliary Lemmas

Lemma 4 (PPO-Clip local smoothness and clipping bias)**.** *Assume A2–A3, the bounded one-step advantage condition A5, and the bounded clipping-probability condition A7. On the event*

$$\mathcal{E}_{\text{clip}} = \{|\rho_{t,i} - 1| \le \varepsilon_{\text{clip}}\},$$

the PPO-Clip surrogate coincides with the unclipped importance-weighted surrogate ρA. Its gradient is Lipschitz with

$$L_{\text{clip}} \le c_0(1 + \varepsilon_{\text{clip}})A_{\max}(L + G^2)$$

for a universal constant c_0. On the complement event $\mathcal{E}^c_{\text{clip}}$, the contribution to the expected gradient is bounded by an additive term of order $\mathcal{O}(\delta_{\text{clip}})$.

Proof. When $|\rho_{t,i} - 1| \le \varepsilon_{\text{clip}}$, we have

$$\text{clip}(\rho_{t,i}, 1 - \varepsilon_{\text{clip}}, 1 + \varepsilon_{\text{clip}}) = \rho_{t,i}.$$

Therefore, on $\mathcal{E}_{\text{clip}}$, the clipped objective locally coincides with the unclipped importance-weighted term $\rho_{t,i}A_{t,i}$.

Differentiating the importance-weighted term gives a gradient involving the policy score function $\nabla_{\theta_i} \log \pi_{\theta_i}(a_{t,i} \mid c_{t,i})$. By A2, the score function is L-Lipschitz, and by A3 its norm is bounded by G. By A5, $|A_{t,i}| \le A_{\max}$. Combining these bounds yields a Lipschitz constant of order

$$(1 + \varepsilon_{\text{clip}})A_{\max}(L + G^2),$$

up to a universal multiplicative constant c_0.

On $\mathcal{E}^c_{\text{clip}}$, bounded score and bounded advantage imply that the gradient contribution is uniformly bounded. Taking expectation and using A7,

$$\Pr(\mathcal{E}^c_{\text{clip}}) \le \delta_{\text{clip}},$$

gives an additive clipping-bias term of order $\mathcal{O}(\delta_{\text{clip}})$. This argument does not require the step size to deterministically keep the likelihood ratio inside the clipping interval. □

Lemma 5 (Verifier-induced Shapley reward properties and Shapley-shaping bias)**.** *Under A1 and Assumption A0, direct verifier-induced Shapley allocation produces per-agent rewards*

$$r^{\text{Shapley}}_{t,i} = \phi_i(v_{q_t,\boldsymbol{a}_t})$$

satisfying

$$\sum_i r^{\text{Shapley}}_{t,i} = \text{Verifier}(q_t, \boldsymbol{a}_t)$$

exactly. Let

$$g^{\text{Sh}}_i = \mathbb{E}[\nabla_{\theta_i} \log \pi_{\theta_i}(a_{t,i} \mid c_{t,i})\, r^{\text{Shapley}}_{t,i}]$$

be the Shapley-shaped local learning signal, and let

$$g^{\text{sys}}_i = \mathbb{E}[\nabla_{\theta_i} \log \pi_{\theta_i}(a_{t,i} \mid c_{t,i})\, \text{Verifier}(q_t, \boldsymbol{a}_t)]$$

be the policy-gradient signal induced by the original system reward. The Shapley-shaping bias for agent i is

$$b_i = g^{\text{Sh}}_i - g^{\text{sys}}_i.$$

Equivalently,

$$b_i = \mathbb{E}[\nabla_{\theta_i} \log \pi_{\theta_i} \cdot (\phi_i(v_{q_t,\boldsymbol{a}_t}) - \text{Verifier}(q_t, \boldsymbol{a}_t))].$$

This bias is generally nonzero: budget balance preserves total reward mass, but it does not imply equality between agent-wise Shapley-shaped learning signals and the policy-gradient signal induced by the original system reward.

Proof. The budget-balance identity follows from Shapley efficiency applied to the coalition game $v_{q_t,\boldsymbol{a}_t}$, whose grand-coalition value is $\text{Verifier}(q_t, \boldsymbol{a}_t)$. The expressions for g^{Sh}_i and g^{sys}_i are the one-step policy-gradient learning signals under the local Shapley reward and the original system reward, respectively. Subtracting them gives the stated bias. The bias need not vanish because $\phi_i(v_{q_t,\boldsymbol{a}_t})$ is an agent-specific redistributed credit, whereas $\text{Verifier}(q_t, \boldsymbol{a}_t)$ is a shared global verifier score. □

Lemma 6 (Coalition-value error to Shapley error)**.** *If*

$$|\hat{v}_\psi(C) - v^{\text{env}}_{q,\boldsymbol{a}}(C)| \le \varepsilon_{\text{est}}$$

for all $C \subseteq \mathcal{N}$, then for every agent i the unnormalized LLM-SHAP *value satisfies*

$$|\hat{\phi}_i - \phi^*_i| \le 2\varepsilon_{\text{est}}.$$

Proof. For each predecessor coalition $C \subseteq \mathcal{N}\backslash\{i\}$, the true marginal contribution is

$$v^{\text{env}}(C \cup \{i\}) - v^{\text{env}}(C),$$

while the learned marginal contribution is

$$\hat{v}_\psi(C \cup \{i\}) - \hat{v}_\psi(C).$$

The difference between these two marginal contributions is bounded by

$$\begin{aligned}&|\hat{v}_\psi(C \cup \{i\}) - v^{\text{env}}(C \cup \{i\})| \\ &\quad + |\hat{v}_\psi(C) - v^{\text{env}}(C)| \\ &\le 2\varepsilon_{\text{est}}.\end{aligned}$$

Since Shapley weights are nonnegative and sum to 1, the weighted average of marginal-contribution errors is also bounded by $2\varepsilon_{\text{est}}$. □

Lemma 7 (Policy-gradient perturbation)**.** *Let the* LLM-SHAP *advantage be*

$$\hat{A}_{t,i} = A^*_{t,i} + \xi_{t,i},$$

where $A^*_{t,i}$ is the advantage induced by the exact Shapley reward. Under A3,

$$\left\|\mathbb{E}[\nabla \log \pi_i \cdot \hat{A}_{t,i}] - \nabla J_{\text{Sh},i}(\theta)\right\| \le G\,\mathbb{E}|\xi_{t,i}|.$$

Under the LLM-SHAP *estimator conditions in A8 and Eq. (7),*

$$\begin{aligned}|\xi_{t,i}| &\le \beta_{\text{ren}} \\ &:= (2 + K_R R_{\max})\varepsilon_{\text{est}} + 2K_R\varepsilon^2_{\text{est}}, \\ K_R &= \frac{2n}{\tau_{\text{floor}}}.\end{aligned}$$

Consequently, the squared perturbation contribution to the convergence bound is

$$\mathcal{O}(K_R^2\varepsilon^2_{\text{est}}),$$

up to constants and higher-order terms. Safeguard steps are budget-balanced but excluded from this closeness-based perturbation bound.

Proof. The first inequality follows from the bounded-score assumption:

$$\begin{aligned}\|\mathbb{E}[\nabla \log \pi_i \cdot \xi_{t,i}]\| &\le \mathbb{E}[\|\nabla \log \pi_i\|\,|\xi_{t,i}|] \\ &\le G\,\mathbb{E}|\xi_{t,i}|.\end{aligned}$$

Under the LLM-SHAP estimator conditions in A8, Theorem 2 gives the bound β_{ren} on the reward perturbation. Squaring the perturbation scale gives the contribution to the gradient-norm floor. The implementation safeguard is not used in this perturbation analysis. □

C.3 Proof of Theorem 1

Proof. For direct verifier-induced Shapley allocation, the grand-coalition value satisfies

$$v_{q_t,\boldsymbol{a}_t}(\mathcal{N}) = \text{Verifier}(q_t, \boldsymbol{a}_t) = R_t.$$

By the efficiency axiom of the Shapley value,

$$\sum_i r^{\text{Shapley}}_{t,i} = \sum_i \phi_i(v_{q_t,\boldsymbol{a}_t}) = v_{q_t,\boldsymbol{a}_t}(\mathcal{N}) = R_t.$$

For LLM-SHAP, if $\hat{\Sigma} \neq 0$, Eq. (7) gives

$$r_{t,i} = R_t\frac{\hat{\phi}_i}{\hat{\Sigma}}, \qquad \hat{\Sigma} = \sum_j \hat{\phi}_j.$$

Therefore,

$$\sum_i r_{t,i} = R_t\frac{\sum_i \hat{\phi}_i}{\hat{\Sigma}} = R_t.$$

If the implementation safeguard is triggered, $r_{t,i} = R_t/n$ also sums to R_t. Thus, both the main normalization rule and the safeguard preserve the realized verifier reward mass. Only the main nondegenerate rescaling rule is used in the estimator-error and convergence analyses. □

C.4 Proof of Theorem 2: Estimator-Error Propagation

Proof. The unnormalized bound

$$|\hat{\phi}_i - \phi^*_i| \le 2\varepsilon_{\text{est}}$$

follows directly from Lemma 6.

It remains to bound the additional distortion introduced by the budget-preserving normalization. Let

$$\hat{\Sigma} = \sum_i \hat{\phi}_i, \qquad \Sigma^* = \sum_i \phi^*_i = R_t.$$

By Lemma 6,

$$|\hat{\Sigma} - \Sigma^*| = \left| \sum_i (\hat{\phi}_i - \phi_i^*) \right| \leq 2n\varepsilon_{\text{est}}.$$

On non-degenerate rescaling steps, A8 gives

$$|\hat{\Sigma}| \geq \tau_{\text{floor}}.$$

The normalized LLM-SHAP reward is

$$r_i = R_t \frac{\hat{\phi}_i}{\hat{\Sigma}}.$$

Therefore,

$$\begin{aligned} |r_i - \phi_i^*| &= \left| R_t \frac{\hat{\phi}_i}{\hat{\Sigma}} - \phi_i^* \right| \\ &= \left| \frac{R_t}{\hat{\Sigma}} \hat{\phi}_i - \hat{\phi}_i + \hat{\phi}_i - \phi_i^* \right| \\ &\leq |\hat{\phi}_i - \phi_i^*| + \left| \frac{R_t - \hat{\Sigma}}{\hat{\Sigma}} \right| |\hat{\phi}_i|. \end{aligned}$$

Using $|R_t - \hat{\Sigma}| \leq 2n\varepsilon_{\text{est}}$ and $|\hat{\Sigma}| \geq \tau_{\text{floor}}$, we obtain

$$\left| \frac{R_t - \hat{\Sigma}}{\hat{\Sigma}} \right| \leq \frac{2n}{\tau_{\text{floor}}} \varepsilon_{\text{est}} = K_R \varepsilon_{\text{est}}.$$

Moreover,

$$|\hat{\phi}_i| \leq |\phi_i^*| + |\hat{\phi}_i - \phi_i^*| \leq R_{\max} + 2\varepsilon_{\text{est}},$$

where $|\phi_i^*| \leq R_{\max}$ follows from bounded verifier rewards and the fact that each Shapley value is a weighted average of marginal contributions between bounded coalition values. Combining the above inequalities gives

$$|r_i - \phi_i^*| \leq 2\varepsilon_{\text{est}} + K_R \varepsilon_{\text{est}} (R_{\max} + 2\varepsilon_{\text{est}}),$$

or equivalently,

$$|r_i - \phi_i^*| \leq (2 + K_R R_{\max}) \varepsilon_{\text{est}} + 2K_R \varepsilon_{\text{est}}^2.$$

If the numerical safeguard $r_{t,i} = R_t/n$ is triggered, the above non-degenerate rescaling argument does not apply; in that case, only budget preservation is claimed. □

C.5 Proof of Theorem 3

Proof. The convergence target is the stacked gradient of the local Shapley-shaped surrogate system,

$$\nabla J_{\text{Sh}} = \big(\nabla_{\theta_1} J_{\text{Sh},1}, \ldots, \nabla_{\theta_n} J_{\text{Sh},n} \big),$$

not the gradient of the original system objective J_{sys}. The proof is stated for the one-step contextual-bandit training regime, where the PPO update uses the one-step action-level advantage estimator in Eq. (8).

Step 1: Smoothness on the clipping region. Applying the standard smooth non-convex descent inequality componentwise to each local surrogate $J_{\text{Sh},i}$ and summing over agents gives a descent inequality for the stacked surrogate system. On the event

$$\mathcal{E}_{\text{clip}} = \{ |\rho_{t,i} - 1| \leq \varepsilon_{\text{clip}} \},$$

Lemma 4 ensures that the clipped surrogate locally coincides with the unclipped importance-weighted surrogate and has Lipschitz gradient. Therefore, for constants $c_1, c_2, c_3, c_4 > 0$,

$$\begin{aligned} &\mathbb{E}[\mathcal{J}_{\text{loc}}(\theta_{k+1})] \\ &\geq \mathbb{E}[\mathcal{J}_{\text{loc}}(\theta_k)] + \eta \mathbb{E} \|\nabla J_{\text{Sh}}(\theta_k)\|^2 \\ &\quad - c_1 \eta^2 - c_2 \varepsilon_{\text{clip}}^2 \\ &\quad - c_3 \delta_{\text{clip}} - c_4 \Delta_{\text{est}}. \end{aligned}$$

Here

$$\mathcal{J}_{\text{loc}}(\theta) = \sum_i J_{\text{Sh},i}(\theta_i; \theta_{-i})$$

is used only as an aggregate Lyapunov function for the proof, and Δ_{est} denotes the reward-estimator perturbation term.

Step 2: Clipping bias. PPO-Clip does not deterministically guarantee $|\rho_{t,i} - 1| \leq \varepsilon_{\text{clip}}$ from the step size alone. We therefore use the bounded clipping-probability condition A7, which bounds the off-clip probability by δ_{clip}. The contribution of off-clip samples to the expected descent inequality is absorbed into the $\mathcal{O}(\delta_{\text{clip}})$ term.

Step 3: Estimator bias. For direct verifier-induced Shapley allocation, coalition values are obtained directly from the verifier, so $\Delta_{\text{est}} = 0$. For LLM-SHAP under the estimator conditions in A8, Lemma 7 gives

$$\Delta_{\text{est}} = K_R^2 \varepsilon_{\text{est}}^2$$

up to constants and higher-order terms. Budget-only safeguard steps are excluded from this closeness-based estimator-bias term.

Step 4: Summation. Summing the descent inequality over $k = 0, \ldots, K-1$, using the bounded local surrogate condition A6, and choosing $\eta = K^{-1/2}$ yields

$$\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\|\nabla J_{\mathrm{Sh}}(\theta_k)\|^2 \le \mathcal{O}(K^{-1/2}) + \mathcal{O}(\varepsilon_{\mathrm{clip}}^2) + \mathcal{O}(\delta_{\mathrm{clip}}) + \mathcal{O}(\Delta_{\mathrm{est}}).$$

This proves the stated convergence bound. The result is a convergence statement for the Shapley-shaped local surrogate system. The gap between this learning signal and the policy-gradient signal induced by the original system objective is characterized by Lemma 5. □

C.6 Auxiliary Micro-Step Markov View

Proposition 8 (Augmented Markov process over micro-steps)**.** *Under the fixed sequential role-based instantiation of Eq. (1), with deterministic transcript-concatenation transitions and a fixed agent ordering, the induced micro-step process admits an MDP on the augmented state*

$$\tilde{s}_{t,i} = (s_t, a_{t,1}, \ldots, a_{t,i-1}, i)$$

indexed by micro-steps (t, i). The one-step action-level advantage estimator in Eq. (8) is defined on this augmented micro-step process for any per-agent reward signal $r_{t,i}$.

Proof. Construct the augmented state

$$\tilde{s}_{t,i} = (s_t, a_{t,1}, \ldots, a_{t,i-1}, i)$$

with augmented action $\tilde{a}_{t,i} = a_{t,i}$. For $i < n$, the augmented transition is

$$\tilde{s}_{t,i+1} = (s_t, a_{t,1}, \ldots, a_{t,i}, i+1),$$

and for $i = n$, the process moves to

$$\tilde{s}_{t+1,1} = (s_{t+1}, 1).$$

Each transition depends only on the current augmented state and the current augmented action. Therefore, the micro-step process is Markov. The text input context $c_{t,i}$ is the prompt-level realization of the augmented micro-step state $\tilde{s}_{t,i}$. The one-step advantage in Eq. (8) is therefore well-defined because both the allocated reward $r_{t,i}$ and the baseline $V_\omega(c_{t,i})$ are indexed by the same micro-step (t, i). □

C.7 Auxiliary Variance Redistribution Result

Proposition 9 (Variance redistribution)**.** *By Shapley efficiency, $\sum_i \phi_i = \mathrm{Verifier}(q, \boldsymbol{a})$. Therefore,*

$$\mathrm{Var}(\mathrm{Verifier}(q, \boldsymbol{a}) \mid q) = \sum_i \mathrm{Var}(\phi_i \mid q) + 2\sum_{i<j}\mathrm{Cov}(\phi_i, \phi_j \mid q).$$

Equal allocation gives every agent the per-agent reward variance $\mathrm{Var}(\mathrm{Verifier}(q, \boldsymbol{a}) \mid q)/n^2$, whereas Shapley allocation gives agent-specific variances $\mathrm{Var}(\phi_i \mid q)$. Thus, Shapley allocation redistributes variance from a uniform per-agent signal to a contribution-dependent signal. The aggregate per-agent variance is lower under Shapley than under Equal if and only if

$$\sum_i \mathbb{E}_q\mathrm{Var}(\phi_i \mid q) < \frac{1}{n}\mathbb{E}_q\mathrm{Var}(\mathrm{Verifier}(q, \boldsymbol{a}) \mid q).$$

Equivalently, using the covariance decomposition above, this condition holds when the cross-agent covariances are sufficiently positive.

C.8 Auxiliary Per-Agent Clipping Diagnostic

Proposition 10 (Clipping-rate diagnostic under a one-step update model)**.** *Under a first-order approximation of one PPO inner update,*

$$\log \rho_{t,i} \approx \eta A_{t,i}\,\|\nabla_{\theta_i}\log \pi_{\theta_i}\|^2,$$

and the bounded-score assumption A3, the clipping probability satisfies

$$\Pr(|\rho_{t,i} - 1| > \varepsilon_{\mathrm{clip}}) \lesssim \frac{\eta^2 G^4}{\log(1+\varepsilon_{\mathrm{clip}})^2}\mathbb{E}[A_{t,i}^2].$$

Thus, per-agent clipping rates are expected to scale with the second moment of the shaped advantage. Under Equal allocation this second moment is tied to the shared reward signal, while under Shapley allocation it becomes agent-specific through ϕ_i.

Proof. Under the stated first-order approximation, the event $|\rho_{t,i} - 1| > \varepsilon_{\mathrm{clip}}$ is controlled by the event

$$|\log \rho_{t,i}| > \log(1 + \varepsilon_{\mathrm{clip}})$$

up to constants. Applying Markov's inequality to the square of the approximated log-ratio and using

$$\|\nabla_{\theta_i}\log \pi_{\theta_i}\| \le G$$

gives the bound. The final interpretation follows by substituting the equal-allocation advantage or the Shapley-shaped advantage into $\mathbb{E}[A_{t,i}^2]$. □

C.9 Auxiliary Conditional Core Guarantee

Definition 11 (Core under policy-snapshot semantics)**.** For a question q and a fixed joint policy snapshot $\boldsymbol{\pi}_{\boldsymbol{\theta}_k}$, define the verifier-induced coalition game

$$v_q(C) := \text{Verifier}(q, \boldsymbol{a}^C).$$

An allocation $\boldsymbol{r} \in \mathbb{R}^n$ lies in the Core of v_q if

$$\sum_{i \in C} r_i \geq v_q(C), \quad \text{for all } C \subseteq \mathcal{N},$$
$$\sum_i r_i = v_q(\mathcal{N}).$$

It lies in the ε-Core if the first inequality holds with slack $-\varepsilon$ and efficiency holds exactly. Core membership is a per-step, policy-snapshot statement; no global Core membership across all training snapshots is claimed.

Theorem 12 (Auxiliary conditional Core guarantee under convex games)**.** *For a fixed task instance and a fixed joint policy snapshot $\boldsymbol{\pi}_{\boldsymbol{\theta}_k}$, define the verifier-induced coalition game*

$$v_q(C) = \text{Verifier}(q, \boldsymbol{a}^C).$$

If v_q is convex, i.e., supermodular over coalitions, then the exact Shapley allocation $\boldsymbol{\phi}^*$ lies in the Core of v_q(Shapley, 1971).

For LLM-SHAP*, under the estimator conditions in A8 and Eq. (7), the allocation satisfies an ε-Core slack whose coalition-wise bound is*

$$\varepsilon_C \leq \big(2|C| + |C|K_R R_{\max}\big)\varepsilon_{\text{est}} + 2|C|K_R\varepsilon_{\text{est}}^2.$$

If the numerical safeguard is triggered, LLM-SHAP *reduces to budget-only Equal allocation; no Core guarantee is claimed beyond budget balance.*

Proof. Step 1: verifier-induced Shapley allocation under convex games. If v_q is convex, every marginal-contribution vector associated with an agent permutation lies in the Core. The Shapley value is the average of these marginal vectors, and the Core of a convex game is convex. Therefore, the exact Shapley allocation $\boldsymbol{\phi}^*$ lies in the Core of v_q.

Step 2: LLM-SHAP *under non-degenerate rescaling.* Let $\hat{\phi}_i$ be the unnormalized Shapley value computed from the learned estimator, and let r_i be the post-normalized reward under Eq. (7). By Lemma 6,

$$|\hat{\phi}_i - \phi_i^*| \leq 2\varepsilon_{\text{est}}.$$

The rescaling analysis in Theorem 2 gives

$$|r_i - \phi_i^*| \leq (2 + K_R R_{\max})\varepsilon_{\text{est}} + 2K_R\varepsilon_{\text{est}}^2.$$

Summing this per-agent deviation over a coalition C gives

$$\varepsilon_C \leq \big(2|C| + |C|K_R R_{\max}\big)\varepsilon_{\text{est}} + 2|C|K_R\varepsilon_{\text{est}}^2.$$

Thus, under non-degenerate rescaling, LLM-SHAP satisfies an ε-Core guarantee with coalition-wise slack ε_C.

Step 3: Numerical safeguard. If the denominator is numerically degenerate, the implementation sets $r_i = R_t/n$, which preserves budget balance. However, Equal allocation need not satisfy coalitional rationality. Therefore, no Core guarantee is claimed for safeguard steps beyond budget balance. □

C.10 Counterexample: Equal Allocation Outside the Core

Consider a 2-agent coalition game with

$$v(\emptyset) = 0, \qquad v(\{1\}) = 1,$$
$$v(\{2\}) = 0, \quad v(\{1,2\}) = 1.1.$$

Core membership requires

$$r_1 \geq 1, \qquad r_2 \geq 0, \qquad r_1 + r_2 = 1.1.$$

Equal allocation gives $r_1 = r_2 = 0.55$, which violates $r_1 \geq 1$, so Equal is outside the Core. Shapley allocation gives

$$\phi_1 = 1.05, \qquad \phi_2 = 0.05,$$

which satisfies the Core constraints. This example only shows that Equal allocation can violate coalitional rationality; it does not claim Pareto domination of Shapley over Equal for every individual agent.

D Algorithmic Details

D.1 Proposed LLM-SHAP 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.

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

Input: Question q_t, grand trajectory $\boldsymbol{a}_t^{\mathcal{N}}$, reward R_t, verifier Verifier
Output: Labeled coalition dataset $\mathcal{D}_t^{\text{coal}}$

$\mathcal{D}_t^{\text{coal}} \leftarrow \emptyset$
for each coalition $C \subseteq \mathcal{N}$ **do**
  if $C = \mathcal{N}$ **then**
    $\boldsymbol{a}_t^C \leftarrow \boldsymbol{a}_t^{\mathcal{N}}$; $y_C \leftarrow R_t$
  else if $C = \emptyset$ or C has no answer-capable agent **then**
    $\boldsymbol{a}_t^C \leftarrow \text{AbsTraj}(q_t, C)$
    $y_C \leftarrow 0$
  else
    $\boldsymbol{a}_t^C \leftarrow \text{CFRollout}(q_t, C)$
    $y_C \leftarrow \text{Verifier}(q_t, \boldsymbol{a}_t^C)$
  $x_C \leftarrow \text{BuildCtx}(q_t, \boldsymbol{a}_t^C, C)$
  $\mathcal{D}_t^{\text{coal}} \leftarrow \mathcal{D}_t^{\text{coal}} \cup \{(C, x_C, y_C)\}$
return $\mathcal{D}_t^{\text{coal}}$

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

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

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

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

Input: Coalition dataset $\mathcal{D}_t^{\text{coal}}$, estimator $\hat{v}_\psi$
Output: Predicted coalition-value table $\widehat{\mathcal{V}}_t$

for each $(C, x_C, y_C) \in \mathcal{D}_t^{\text{coal}}$ **do**
  $\widehat{\mathcal{V}}_t(C) \leftarrow \hat{v}_\psi(x_C)$
return $\widehat{\mathcal{V}}_t$

Algorithm 5 FitV: estimator fitting before estimator freezing.

Input: Coalition dataset $\mathcal{D}_t^{\text{coal}}$, estimator $\hat{v}_\psi$
Output: Updated estimator parameters ψ

$\mathcal{L}_\psi \leftarrow 0$
for each $(C, x_C, y_C) \in \mathcal{D}_t^{\text{coal}}$ with $y_C \neq$ None **do**
  $\mathcal{L}_\psi \leftarrow \mathcal{L}_\psi + (\hat{v}_\psi(x_C) - y_C)^2$
Update ψ by minimizing $\mathcal{L}_\psi$
return ψ

D.2 LLM-SHAP Shapley-Value Allocator

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 3.1. 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. The grand coalition reuses the cached verifier reward R_t, 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 $\sum_i \phi_i = v^{\text{env}}(\mathcal{N}) = R_t$.

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. We do not treat these estimates as alternative verifier scores or direct evidence of hidden rewards. They are used as Shapley-shaped learning signals whose reliability is assessed through mechanism diagnostics.

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.

Algorithm 6 ALLOC: Shapley allocation with reward normalization.

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

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

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

Input: Question q_t, grand coalition trajectory $\boldsymbol{a}_t$, reward R_t, verifier Verifier
Output: Verifier-induced coalition-value table v^{env}

for each coalition $C \subseteq \mathcal{N}$ **do**
  if $C = \mathcal{N}$ **then**
    $v^{\text{env}}(C) \leftarrow R_t$
  else if $C = \emptyset$ or C has no answer-capable agent **then**
    $v^{\text{env}}(C) \leftarrow 0$
  else
    $\boldsymbol{a}_t^C \leftarrow \text{COUNTERFACTUALROLLOUT}(q_t, C)$
    $v^{\text{env}}(C) \leftarrow \text{Verifier}(q_t, \boldsymbol{a}_t^C)$
return v^{env}

E Implementation Details

This appendix provides the implementation details used in the main experiments and the core ablations. Unless explicitly stated otherwise, all compared methods use the same task split, backbone model, agent profiles, LoRA configuration, optimizer settings, environment-step rollout budget, decoding setup, random seeds, checkpoint-selection rule, and evaluation protocol within the same task family. The controlled comparison keeps the multi-agent fine-tuning scaffold fixed and varies the credit-construction mechanism used to derive agent-level rewards from verifier feedback.

E.1 Fair Comparison Protocol

For a controlled comparison, MARFT-A, C3, CCPO, SHARP, the verifier-induced Shapley ablation, and the LLM-SHAP algorithm are trained under matched optimization settings, environment-step rollout budgets, checkpoint-selection rules, and evaluation protocols. Within each task family, all methods use the same backbone, role-specific LoRA configuration, agent prompts, dataset split, maximum generation length, number of environment steps, PPO hyperparameters, and evaluation metric.

We run each method with five random seeds and report mean $\pm$ standard deviation. The seed identifiers are: $3, 13, 23, 33, 43$. For each run, we report the best checkpoint selected by performance on the corresponding evaluation benchmark. The same checkpoint-selection rule is used for all compared methods. If a validation set is used for checkpoint selection, it is not used for final test reporting.

Hyperparameters are fixed before comparing credit-assignment methods. We do not tune a separate hyperparameter set for each baseline. If a method requires a method-specific hyperparameter, we use the value recommended by the original paper when applicable; otherwise, we use the same normalization floor and safeguard rule as our allocation layer.

E.2 Effect-Size Context

Effect-size context. The gains in Table 1 should be interpreted as controlled reward-allocation gains rather than unconstrained model-scaling gains. All compared fine-tuning methods share the same multi-agent scaffold, backbone family, LoRA configuration, prompts, optimizer settings, environment-step budget, checkpoint-selection rule, and evaluation protocol; they differ only in how system-level verifier feedback is converted into agent-level training rewards.

Compared with the strongest non-MASHAP-LLM baseline in each matched setting, MASHAP-LLM achieves average gains of 2.51 absolute points under the dual-agent setting and 4.36 absolute points under the triple-agent setting. On mathematical reasoning, the absolute gains range from 1.71 to 7.19 points, while on Codeforces the gains are 1.05 and 1.64 points under the dual-agent and triple-agent settings, respectively. The largest relative margins over the strongest non-MASHAP-LLM baseline reach 15.1% on CMATH→MATH and 34.0% on Codeforces.

Interpretation. These margins are meaningful because they are obtained under matched training and evaluation settings, where the primary experimental variable is the credit-construction mechanism. The larger gains in several triple-agent settings are consistent with the motivation of MASHAP-LLM: when agents play heterogeneous roles and contribute unequally, LLM-SHAP provides a learned coalition-value game that yields more informative per-agent learning signals than assigning identical global credit. We report these effect-size summaries to contextualize the internal

controlled comparison, rather than to make direct numerical comparisons with methods trained under different backbones, supervision recipes, reward designs, or training protocols.

E.3 Task Setup and Evaluation

Table 4 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 a held-out Codeforces test split.

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 5: LoRA and adapter configuration.

E.5 Rollout and Generation Settings

Table 6 reports the rollout and generation settings used for mathematical reasoning and Codeforces experiments. Optimization-specific hyperparameters are reported separately in Table 7.

Setting	Math	Codeforces
Rollout threads	1	4
Max horizon	2	1
Episode length	8	30
Context window	2,456	4,096
Max new tokens	512	1,024

Table 6: Rollout and generation settings.

E.6 Optimization Hyperparameters

Table 7 reports the core PPO optimization hyperparameters. These settings are shared across MARFT-A, C3/CCCAMA, CCPO, verifier-induced Shapley allocation, and the LLM-SHAP algorithm unless explicitly stated otherwise.

Hyperparameter	Math	Codeforces
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 7: Core PPO optimization hyperparameters. LLM-SHAP estimator-specific settings are reported separately in Table 8.

E.7 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. In our target regime, the number of agents is small ($n \leq 4$), so all 2^n coalitions are enumerated exactly during supervised fitting and all 2^n coalition masks are enumerated exactly during frozen inference.

Item	Value
Coalition enumeration	Exact enumeration of all 2^n coalitions during supervised fitting and all 2^n masks during frozen inference
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 8: LLM-SHAP estimator and reward-allocation settings.

For mathematical reasoning, the LLM-SHAP estimator uses Qwen2.5-Coder-3B-Instruct; for

Item	Mathematical Reasoning	Code Generation
Backbone	Qwen2.5-Coder-3B-Instruct	Qwen2.5-3B-Instruct
Training data	MATH / CMATH	Codeforces
Evaluation data	MATH500, CMATH, GSM8K	Held-out Codeforces
Dataset size	MATH: 7,500 / 500 CMATH: 600 / 1,098 GSM8K: eval-only, 1,319	Codeforces: 6,610 / 393
Data source	Public mathematical reasoning benchmarks	Public Codeforces programming problems
Split protocol	Official splits when available	Processed verifiable split
Generation	Greedy decoding	Greedy decoding
Evaluation metric	Pass@1 accuracy	Unit-test pass rate
Answer extraction / execution	Boxed-answer extraction	Compilation and execution
Correctness check	Exact match	Unit tests
Checkpoint selection	Validation-best checkpoint	Validation-best checkpoint

Table 4: 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 is used only for evaluation. Codeforces uses a processed verifiable split from public programming problems and evaluates generated programs by compilation and unit tests.

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.7.1 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

$$\mathcal{L}^{(m)}_{\text{val}} = \frac{1}{|\mathcal{D}^{\text{val}}_{\text{coal}}|} \sum_{(x_C, y_C) \in \mathcal{D}^{\text{val}}_{\text{coal}}} \left(\hat{v}_{\psi_m}(x_C) - y_C\right)^2. \tag{10}$$

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

$$\text{Imp}^{(m)} = \frac{\mathcal{L}_{\text{best}} - \mathcal{L}^{(m)}_{\text{val}}}{\max(\mathcal{L}_{\text{best}}, \epsilon_{\text{loss}})}. \tag{11}$$

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 $x^{\text{infer}}_{t,C}$ for estimator inference, but it does not generate $\boldsymbol{a}^C_t$ 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_t for budget preservation. Importantly, R_t 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 MASHAP-LLM update 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.

We monitor estimator-fidelity diagnostics on logged coalition contexts, including MAE_v,

Variant	Env-CF	Time	Score
Equal / MARFT-A	–	1.00×	46.00
Static-masked	–	1.05×	47.20
Verifier-induced	+	1.45×	49.60
MASHAP-LLM / LLM-SHAP	early	1.65×	54.47

Table 9: Numerical summary of the performance–cost trade-off. Time is normalized by Equal / MARFT-A; "early" indicates environment-level counterfactual rollout used before the estimator is frozen.

RMSE_v, and MAE_ϕ, to verify that the frozen estimator remains accurate enough for allocation.

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.8 Performance–Cost Trade-off

Exact coalition-value acquisition introduces additional training-time overhead before estimator freezing because it requires coalition-conditioned rollouts and verifier evaluation for supervised estimator fitting before applying the Shapley allocator. After freezing, allocation keeps exact mask enumeration but replaces subcoalition rollouts and verifier calls with batched LLM-SHAP forward predictions. To quantify this overhead, we report a performance–cost trade-off under the representative $N = 3$ setting. Training time is normalized by Equal / MARFT-A under the same task, backbone, agent configuration, optimizer, decoding setup, and training budget.

Table 9 show that counterfactual-rollout-based coalition-value supervision increases training time before the estimator-freezing rule is satisfied, but the additional cost is associated with stronger final performance. Static-masked Shapley avoids coalition-specific re-rollout and is therefore cheaper, but it performs substantially worse than counterfactual-rollout-based Shapley. Verifier-induced Shapley improves over static masking, indicating that coalition-specific rollout provides more meaningful reference coalition values. MASHAP-LLM with LLM-SHAP obtains the best final performance, suggesting that learned contextual coalition-value estimation provides additional benefit beyond direct verifier-scored coalition values, while the frozen allocation phase avoids subcoalition environment rollouts and verifier evaluations.

The exact implementation is intended for small role-specialized teams. Exact Shapley allocation requires constructing a coalition-value table over 2^n coalitions, which is manageable in the $N = 2$ and $N = 3$ settings studied here but becomes expensive as n grows. Scaling beyond this regime requires approximate Shapley estimation, structured coalition pruning, or learned coalition sampling.

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 prompts under the same task and agent-count setting; they differ only in the credit-assignment rule used during reinforcement fine-tuning. 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 10: 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 11: 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 → Verifier

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

Table 12: Three-agent mathematical reasoning pipeline.

Math 3-Agent Reasoner: system prompt

```
<|im_start|>system: 3 LLM agents (
    Reasoner -> Actor -> Verifier)
    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 3-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 3-Agent Actor: system prompt

```
<|im_start|>system: 3 LLM agents (
    Reasoner -> Actor -> Verifier)
    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 3-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.
```

Math 3-Agent Verifier: system prompt

```
<|im_start|>system: 3 LLM agents (
    Reasoner -> Actor -> Verifier)
    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 **Verifier**. Review
    the Actor's result against the
    original problem and the
    Reasoner's plan. Check algebra,
     logic, domains, theorem
    assumptions, and whether the
    answer satisfies the problem
    statement. Use substitution,
    alternative reasoning, or
    sanity checks when possible. If
     an inconsistency appears,
    clearly request rethinking and
    specify the failed step.
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 3-Agent Verifier: user prompt template

```
Question:
{question}

Previous agent messages:
{history}

Please verify the Actor's solution
     against the original problem.
    Check algebra, logic, domains,
    edge cases, and final-answer
    validity. If an inconsistency
    is found, identify the failed
    step and request revision.
```

F.2.2 Codeforces: Reasoner → Coder → Reviewer

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

Table 13: 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 LLM-SHAP 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.

We therefore report a lightweight likelihood-based diagnostic. For each sampled coalition intervention, we compute the token-normalized negative log-likelihood (NLL) of the next active agent's continuation under two matched contexts: the reference context and the absence-message context. We define

$$\begin{aligned} \Delta\mathrm{NLL}_{\mathrm{next}} &= \mathrm{NLL}_{\mathrm{abs}} - \mathrm{NLL}_{\mathrm{ref}}, \\ \mathrm{NLL}_{\mathrm{abs}} &= \mathrm{NLL}(a_{\mathrm{next}} \mid c_{\mathrm{abs}}), \\ \mathrm{NLL}_{\mathrm{ref}} &= \mathrm{NLL}(a_{\mathrm{next}} \mid c_{\mathrm{ref}}). \end{aligned} \tag{12}$$

Here c_{abs} denotes the context containing the fixed absence message, and c_{ref} denotes the matched reference context.

Table 14 shows that the downstream NLL shift remains small across all evaluated settings, ranging from 0.01 to 0.04. This indicates that the fixed absence message does not substantially reduce the likelihood of the next active agent's continuation under the frozen backbone. Thus, while absence-message intervention remains a potential source of textual distribution shift, the observed likelihood shift is limited in our evaluated settings.

Setting	$\Delta\mathrm{NLL}_{\mathrm{next}}$
MATH, $N = 2$	0.02
MATH, $N = 3$	0.01
CMATH, $N = 2$	0.03
CMATH, $N = 3$	0.02
Codeforces, $N = 3$	0.04

Table 14: Absence-message NLL diagnostic. Lower values indicate weaker downstream likelihood shift.

G Qualitative Example

A 3-agent team (Reasoner, Solver, Verifier) on a MATH problem of difficulty 4. *Note:* the coalition values $v(C)$ in Table 15 are illustrative, taken from a hypothetical partial-credit verifier; under our actual binary verifier, $v(\mathcal{N}) \in \{0, 1\}$ and $v(C) \in \{0, R\}$ for non-empty answer-capable coalitions. We use partial-credit values purely for pedagogical clarity.

Problem: The hyperbola $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$ has focal length $2c$ with $c = 5$ and asymptote $y = \frac{4}{3}x$. Compute the eccentricity.

Reasoner: "Asymptote slope $b/a = 4/3$, so $b = 4k$, $a = 3k$. With $c^2 = a^2 + b^2$: $25 = 25k^2$ so $k = 1$."

Solver: "$a = 3$, $c = 5$, eccentricity $e = c/a = 5/3$."

Verifier: "Check $e > 1$ ✓; $b/a = 4/3$ ✓; $a^2 + b^2 = 25 = c^2$ ✓. Final answer $\boxed{5/3}$."

Coalition C	$v(C)$
$\emptyset$	0.00
$\{1\}$	0.00 (no answer)
$\{2\}$	0.30 (partial)
$\{3\}$	0.10 (verify only)
$\{1, 2\}$	0.85
$\{1, 3\}$	0.20
$\{2, 3\}$	0.65
$\{1, 2, 3\}$	1.00
ϕ (Shapley)	(0.225, 0.600, 0.175)

Table 15: Illustrative coalition values and Shapley allocation. Equal would give $(0.333, 0.333, 0.333)$, under-crediting the Solver. The state-induced game is *not* convex (the Solver's marginal contribution to $\{1\}$ exceeds its contribution to $\{1, 3\}$), so Theorem 12 does not guarantee Core membership for this state; the example illustrates Shapley behavior, not Core properties.

H Additional Analysis

LLM-SHAP as the proposed allocation algorithm. The empirical results reported in Table 1 correspond to the early-fitted and then frozen LLM-SHAP 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.

Non-Actor underestimation. Direct verifier-induced Shapley allocation may underestimate agents that do not directly produce the final answer, such as Reasoner-only or Verifier-only coalitions. When the verifier only recognizes final-answer success, a Reasoner-only or Verifier-only coalition can have $v(C) = 0$ and contribute little or nothing to the marginal sums, so the corresponding Shapley credit is structurally small. LLM-SHAP can soften this issue 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 central reason the inference rule of Eq. (7) avoids hard-coding such coalitions to zero. This effect motivates the diagnostics in Appendix J, which evaluate whether the learned coalition-value estimator accurately approximates verifier-induced coalition values from frozen masked contextual predictions.

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 $v(C) \in [0, 1]$. This increases the frequency of differentiating episodes in which direct verifier-induced Shapley allocation differs from Equal allocation.

Consistent with this, the LLM-SHAP allocation algorithm 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 1).

Static-masked Shapley, SHARP, and agent-count considerations. The static-masked Shapley variant is included in the ablation study (Table 2). 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. The degradation under static-masked Shapley shows that coalition-specific counterfactual rollout is important for constructing meaningful coalition values.

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 2,

Method	MATH →MATH	MATH →CMATH	MATH →GSM8K
Single-Agent SFT	35.49 ± 1.43	79.24 ± 0.84	66.53 ± 1.18
Single-Agent PPO	36.44 ± 1.35	80.24 ± 0.62	68.45 ± 1.68

Table 16: Single-agent RL control under the MATH-trained setting. Single-Agent SFT denotes direct inference without reinforcement fine-tuning. Single-Agent PPO tests whether verifier-based reinforcement fine-tuning with a direct solver is already sufficient under the same training and evaluation protocol. This experiment is an architecture-control analysis rather than a credit-assignment comparison, since multi-agent reward allocation is not defined for $N = 1$.

with implementation details in Appendix D.4.

I Single-Agent RL Control

We additionally report a single-agent PPO control on the MATH-trained setting to examine whether verifier-based reinforcement fine-tuning with a direct solver is already sufficient for mathematical reasoning tasks. This experiment is not a reward-allocation baseline, because agent-level credit assignment is not defined when there is only one agent. Instead, it serves as an architecture-control experiment: it 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 16 is intended to rule out the alternative explanation that the gains observed in the main MATH-trained experiments can be fully explained by applying verifier-based RL to a single direct solver. If single-agent PPO improves over the single-agent SFT baseline but remains below the multi-agent results reported in Table 1, then single-agent verifier-based RL is useful but insufficient to explain the gains from role-specialized multi-agent fine-tuning. The main credit-assignment evidence remains the controlled $N = 2$ and $N = 3$ multi-agent comparison in Table 1, where the compared methods share the same multi-agent topology and differ in how the system-level verifier reward is converted into agent-level training signals.

J 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) whether budget-preserving normalization is stable and rarely falls back to equal allocation; and (iv) whether LLM-SHAP produces role-differentiated rewards more frequently than direct verifier-induced Shapley under sparse feedback, without using subcoalition rollouts after freezing.

J.1 Estimator Fidelity

Since LLM-SHAP is trained as a text-to-value regression model, we first evaluate estimator fidelity. We report three diagnostics: MAE_v, RMSE_v, and MAE_ϕ. MAE_v and RMSE_v measure the prediction error between learned coalition values $\hat{v}(C)$ produced from frozen mask-conditioned contexts and verifier-induced coalition values $v^{\text{env}}(C)$ collected as reference labels. MAE_ϕ measures the difference between Shapley credits induced by the learned coalition-value table and those induced by the verifier-induced reference game. Thus, MAE_v and RMSE_v evaluate coalition-value prediction quality, while MAE_ϕ evaluates whether value-prediction errors materially distort the downstream allocation.

Setting	$\text{MAE}_v \downarrow$	$\text{RMSE}_v \downarrow$	$\text{MAE}_\phi \downarrow$
MATH, $N = 2$	0.048	0.063	0.039
MATH, $N = 3$	0.052	0.068	0.043
CMATH, $N = 2$	0.050	0.065	0.041
CMATH, $N = 3$	0.054	0.071	0.046
Code, $N = 2$	0.049	0.066	0.042
Code, $N = 3$	0.051	0.069	0.045

Table 17: Estimator-fidelity diagnostics for the early-fitted and frozen LLM-SHAP protocol. Lower values indicate better coalition-value prediction and more faithful Shapley allocation.

Interpretation. Table 17 shows that the frozen learned estimator maintains low coalition-value prediction error across training settings. Low MAE_v indicates accurate average prediction, while low RMSE_v suggests that large prediction failures are uncommon. Low MAE_ϕ further shows that prediction errors do not substantially distort the downstream Shapley allocation. These diagnostics support the use of the frozen estimator for allocation-time coalition-value prediction from mask-conditioned contexts after estimator freezing.

Variant	$\text{MAE}_v \downarrow$	$\text{RMSE}_v \downarrow$	$\text{MAE}_\phi \downarrow$
Frozen LLM-SHAP	0.048	0.063	0.039
Same-step fit	0.056	0.074	0.047
Delayed LLM-SHAP	0.061	0.081	0.052
Hard-zero infer.	0.050	0.066	0.061

Table 18: Internal diagnostics of LLM-SHAP variants. "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.

J.2 Internal Estimator Variants

We further analyze internal design choices of the learned LLM-SHAP estimator. While Table 2 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 the diagnostics on logged grand-coalition trajectories and verifier-derived reference labels without re-running reinforcement fine-tuning for each variant. For each rollout step t and coalition $C \subseteq \mathcal{N}$, each variant produces an allocation-time coalition-value prediction $\hat{v}(C)$ from the same mask-conditioned context $x_{t,C}^{\text{infer}}$ unless the diagnostic explicitly studies a rollout-context variant. We compare $\hat{v}(C)$ with the verifier-induced coalition value $v_t^{\text{env}}(C)$, and compare the resulting Shapley credits $\hat{\phi}_{t,i}$ with the verifier-induced Shapley credits $\phi_{t,i}^{\text{env}}$.

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 fitted diagnostic** updates the value head on the current step's labels before prediction and is reported only as an internal diagnostic, not as the main training protocol. **Delayed LLM-SHAP** uses a lagged estimator checkpoint for allocation. **Hard-zero inference** clamps coalitions without answer-capable agents to zero at inference time before applying the Shapley formula.

Interpretation. Table 18 shows that the proposed early-fitted and frozen LLM-SHAP protocol maintains low estimator and Shapley-allocation errors after estimator freezing. The same-step fitted diagnostic is reported only to test whether fitting on the current labels before prediction changes estimator behavior; it is not used as the main allocation protocol. Delayed estimation increases error because it uses older coalition evidence, while hard-zero inference increases MAE_ϕ because clamping no-answer-capable coalitions changes the marginal-contribution structure before Shapley allocation.

J.3 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. The purpose of these diagnostics is not to introduce new evaluation metrics, but to verify that frozen LLM-SHAP produces non-degenerate and role-sensitive credit signals from masked contextual predictions.

Diagnostic metrics. We report five diagnostic quantities. First, we measure the magnitude of the predicted Shapley sum,

$$\hat{\Sigma}_t = \sum_{i=1}^{n} \hat{\phi}_{t,i},$$

which determines the stability of budget-preserving normalization. A non-degenerate $\hat{\Sigma}_t$ indicates that LLM-SHAP produces a meaningful learned coalition-value game rather than relying on numerical fallback.

Second, we report the safeguard trigger rate:

$$\text{Safeguard} = \frac{1}{|\mathcal{D}|} \sum_{t \in \mathcal{D}} \mathbf{1}\{|\hat{\Sigma}_t| < \tau_{\text{floor}}\}.$$

A low safeguard rate shows that LLM-SHAP rarely falls back to equal allocation.

Third, we report coalition-value prediction error, measured by the mean absolute error between the predicted coalition value and the verifier-derived coalition target collected for supervision or offline diagnostics:

$$\text{MAE} = \frac{1}{|\mathcal{D}|\, 2^n} \sum_{t \in \mathcal{D}} \sum_{C \subseteq \mathcal{N}} \left| \hat{v}_{\psi,t}(C) - y_{t,C} \right|.$$

This evaluates whether the learned estimator captures useful coalition-level information.

Fourth, we measure the verifier-induced differentiating rate. An episode is verifier-differentiating if at least one strict subcoalition receives a verifier score different from the grand coalition. This

quantity reflects how often direct verifier-induced Shapley allocation can differ from equal allocation under sparse binary feedback.

Finally, we report the LLM-SHAP differentiating rate. An episode is LLM-SHAP-differentiating if the normalized LLM-SHAP reward vector differs from equal allocation by more than a small tolerance:

$$\frac{\left\| \boldsymbol{r}_t^{\text{LLM-SHAP}} - R_t \mathbf{1}/n \right\|_1}{R_t + \epsilon} > \epsilon_{\text{diff}}.$$

In our implementation, we use $\epsilon_{\text{diff}} = 0.1$ and a small numerical constant ϵ.

Interpretation. Table 19 shows that LLM-SHAP produces stable and non-degenerate coalition-value estimates across training settings. The predicted Shapley sum $|\hat{\Sigma}|$ remains well above the normalization threshold for most episodes, and the safeguard trigger rate is consistently low. This indicates that the gains of LLM-SHAP are not caused by frequent fallback to equal allocation. The low estimator error further suggests that the mask-conditioned grand-trajectory context contains useful information for predicting verifier-derived coalition values.

The differentiating-rate diagnostics further explain the advantage of LLM-SHAP under sparse binary feedback. Direct verifier-induced Shapley allocation can differentiate agents only when strict subcoalitions receive different verifier scores. By contrast, LLM-SHAP produces role-differentiated reward vectors in a larger fraction of episodes, providing denser credit signals even when the final verifier reward is sparse. These diagnostics indicate that frozen LLM-SHAP provides stable and role-differentiated allocation signals under sparse verifier feedback.

J.4 Role-Wise Credit Distribution

Finally, we examine whether the learned allocation collapses to uniform credit or answer-only credit. Table 20 reports the average normalized reward share assigned by LLM-SHAP to each role on successful episodes:

$$\bar{c}_i = \mathbb{E}\left[\frac{r_{t,i}^{\text{LLM-SHAP}}}{R_t + \epsilon} \;\middle|\; R_t > 0 \right].$$

Setting	Reasoner	Actor	Verifier
MATH-trained	0.25	0.55	0.20
CMATH-trained	0.20	0.60	0.20
Codeforces-trained	0.26	0.60	0.14

Table 20: Average normalized LLM-SHAP credit share on successful episodes. LLM-SHAP assigns the largest reward share to the answer-producing Actor while preserving non-trivial credit for reasoning and verification roles, indicating that the learned reward allocation neither collapses to answer-only credit nor reduces to uniform allocation.

Interpretation. The Actor receives the largest credit share, which is expected because it is the answer-producing agent. However, the Reasoner and Verifier also receive consistent non-zero reward mass. This pattern indicates that LLM-SHAP captures latent support contributions from non-answer agents, rather than simply assigning all credit to the final answer generator. Together with the differentiating-rate diagnostics, this result supports the interpretation that the learned estimator supplies denser role-aware shaping signals under sparse verifier feedback.

K Seed-Level Reliability Analysis

To assess whether the main gains are robust to random-seed variation, we compare MASHAP-LLM with the strongest non-MASHAP-LLM baseline under matched settings using the same five random seeds as Table 1. In our experiments, C3 is the strongest non-MASHAP-LLM baseline in most settings, while SHARP is close to Equal allocation and weaker than CCPO and C3. We therefore report paired seed-level gaps against the strongest non-MASHAP-LLM comparator.

For each seed s, the paired gap is

$$d_s = x_s^{\text{MASHAP-LLM}} - x_s^b, \tag{13}$$

where x_s^b denotes the result of the strongest non-MASHAP-LLM baseline under the same seed, training set, evaluation benchmark, and agent configuration.

Interpretation. The paired seed-level analysis provides additional reliability evidence for the main comparisons. Across matched settings, MASHAP-LLM generally maintains positive paired gaps against the strongest non-MASHAP-LLM baseline and wins on most random seeds. Larger-margin settings further show confidence intervals

Setting	$\lvert\hat{\Sigma}\rvert$ Mean	$\lvert\hat{\Sigma}\rvert$ P5	Safeguard Rate ↓	Estimator MAE ↓	Verifier-Diff.	LLM-SHAP-Diff. ↑
MATH-trained, $N = 2$	0.81	0.20	0.8%	0.048	10%	60%
MATH-trained, $N = 3$	0.80	0.20	1.1%	0.052	14%	70%
CMATH-trained, $N = 2$	0.79	0.21	0.9%	0.050	12%	75%
CMATH-trained, $N = 3$	0.81	0.25	1.3%	0.054	15%	66%
Codeforces-trained, $N = 2$	0.80	0.20	1.0%	0.049	10%	76%

Table 19: Allocation-stability and differentiating-rate diagnostics for frozen LLM-SHAP. The predicted Shapley sum remains non-degenerate, the equal-allocation safeguard is rarely triggered, and the estimator achieves low coalition-value prediction error from masked contexts. Moreover, LLM-SHAP produces role-differentiated rewards more frequently than direct verifier-induced differentiating events under sparse binary feedback.

above zero, indicating that the observed gains are unlikely to be caused by a single favorable seed.

Setting	**Δ**	**95% CI**	**W**	p
Dual-agent setting ($N = 2$)				
M→M	3.70	[0.95, 6.45]	5/5	0.018
M→C	2.39	[0.20, 4.58]	4/5	0.034
M→G	2.10	[0.60, 3.60]	5/5	0.016
C→M	3.20	[0.79, 5.61]	5/5	0.019
C→C	1.71	[-0.47, 3.89]	4/5	0.086
C→G	3.43	[1.09, 5.77]	5/5	0.014
Code	1.05	[-0.54, 2.64]	4/5	0.132
Triple-agent setting ($N = 3$)				
M→M	6.32	[2.88, 9.76]	5/5	0.007
M→C	4.61	[1.54, 7.68]	5/5	0.014
M→G	3.68	[-1.75, 8.71]	4/5	0.112
C→M	7.19	[4.09, 10.29]	5/5	0.003
C→C	2.67	[0.45, 4.89]	4/5	0.029
C→G	4.44	[0.29, 8.59]	4/5	0.041
Code	1.64	[-0.06, 3.44]	4/5	0.058

Table 21: Seed-level reliability against the strongest non-MASHAP-LLM baseline. Δ is the paired gap over five matched seeds, W denotes win seeds, and the 95% CI is computed over matched seed-level differences. The p value is reported as exploratory reliability evidence rather than as a universal significance claim.