AGENTS THAT MATTER: HOW SUBTLE CHOICES SHAPE AGENT ATTRIBUTION IN MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

As multi-agent systems (MAS) become increasingly complex, identifying the contributions of agents is critical for system optimization. However, existing approaches lack a unified formulation for credit assignment. In this work, we formalize agent attribution as a cooperative game, parameterized by the coalition distribution, intervention protocol, and target metric. Using this framework, we demonstrate that intervention protocols induce distinct games: Agent ablation isolates structural bottlenecks, whereas introspective LLM judges fail to faithfully approximate this behavior. We further find that Leave-One-Out (LOO) identifies bottleneck agents as effectively as combinatorial methods at much lower computational cost. Treating prompt optimization as an intervention protocol, we show that prompt-conditioned attribution can identify optimization targets and improve MAS performance. Finally, we apply our framework to audit a medical MAS, revealing that agent contributions to diagnostic accuracy and ethical behavior are often decoupled. By intervening on counterproductive roles, we observe an increase in ethics alignment while maintaining diagnostic accuracy. Overall, this work provides a principled approach for cost-effective MAS attribution and intervention. Code for reproducing our experiments is available at `https://anonymous.4open.science/r/agent-eval-B850`.

1 INTRODUCTION

Large language models (LLMs) are increasingly deployed as interactive agents (Wu et al., 2024), enabling collaborative multi-agent systems (MAS) that tackle complex tasks such as software engineering (Hong et al., 2023; Qian et al., 2024) and medical diagnosis (Chen et al., 2025a). Yet as these systems become more capable and complex, their performance becomes harder to trace back to specific agents: An agent's contribution depends not only on its underlying model capability but also on its assigned role and interactions with other agents (Zhu et al., 2025). This creates a challenge for MAS evaluation and optimization: *How can we quantify each agent's contribution to the overall system performance and identify bottleneck agents?* Addressing this challenge is critical for diagnosing and improving multi-agent LLM systems (Liu et al., 2023a;c).

To address this challenge, recent works adopt methods based on cooperative game theory, which measure marginal contributions by re-evaluating the MAS under different agent coalitions—subsets formed by sampled agents. For example, Cui et al. (2025) use leave-one-out (LOO) attribution, while others (Xia et al., 2026; Ebrahimi et al., 2025) estimate Shapley values (Shapley et al., 1953; Lundberg & Lee, 2017) over multiple coalitions. However, evaluating many coalitions can be computationally expensive, and ablating structurally essential agents, such as orchestrators, can make the system non-executable. Existing methods address these issues in different ways, including replacing agent ablation with in-context simulation (Cui et al., 2025), replacing an agent's backbone model rather than removing it (Yang et al., 2025), or restricting coalitions according to system topology (Xia et al., 2026). Yet it remains unclear how these choices—which coalitions are considered and how agents are intervened on—shape attribution across MAS structures, leaving the trade-offs among attribution methods and their suitability for different interventions poorly understood.

To enable this investigation, we formalize agent attribution as an *intervention-conditioned cooperative game* (Covert et al., 2021). Under this formulation, an attribution query is parameterized by three variables: (i) the intervention protocol, (ii) the coalition distribution, and (iii) the target metric. Attribution scores are therefore defined relative to an evaluation query, rather than as a universal

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

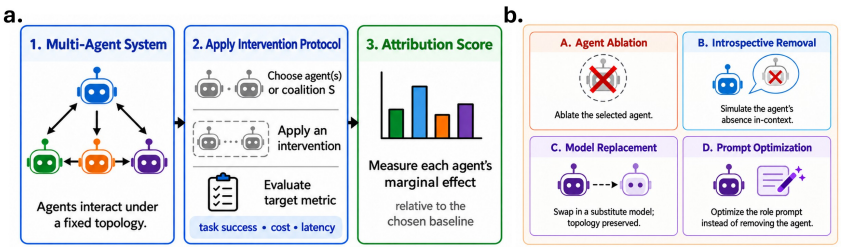


Figure 1: **(a)** Overview of intervention-based attribution for MAS and **(b)** intervention protocols.

importance score. Using this framework, our analysis reveals several key insights. First, distinct intervention approaches induce different attribution games: Agent ablation concentrates attribution on structurally indispensable roles, e.g., orchestrator, and introspective removal with an LLM judge is not faithful to this ablation. Second, despite its simplicity, LOO effectively identifies key agents while reducing computational costs by $2.2\times$–$11.9\times$ compared to methods that require combinatorial coalitions across datasets. Under our framework, prompt optimization defines a distinct intervention-conditioned attribution game. Using prompt-optimization LOO, we identify effective optimization targets while reducing attribution cost by 12% relative to prior work (Xia et al., 2026) and using 79% fewer tokens than prompt-optimization Shapley. Finally, we show that agent importance is objective-dependent. When auditing a medical MAS, we find that agent contributions to diagnostic accuracy and ethical alignment are often decoupled; agents that improve accuracy may actively degrade ethics scores. Overall, we establish a unified framework for agent attribution, providing a principled and practical basis for selecting attribution methods and guiding MAS interventions.

2 Related Works

Instance-level attribution and trace-based diagnosis. Liu et al. (2023c) propagate peer ratings through communication graphs, while Cui et al. (2025) approximate leave-one-out removal through in-context simulation. Failure-diagnosis methods localize errors to agents and steps (Zhang et al., 2025) or characterize recurring interaction failures (Cemri et al., 2026). DoVer (Ma et al., 2026) goes beyond trace inspection by testing message and plan interventions through re-execution. Our focus is complementary to instance-specific repair: estimating intervention effects across a task distribution to guide system-level intervention.

Credit assignment for agent coordination. Beyond post-hoc analysis, contribution estimates can also shape agent interactions during execution. Ebrahimi et al. (2025) dynamically update agent credibility for adversary-resistant aggregation, while Shapley-Coop (Hua et al., 2026) uses Shapley-based credit assignment for negotiation and reward redistribution. These methods use attribution to coordinate agent behavior during execution; we instead study how the underlying game formulation shapes the distribution of attribution values.

Global agent attribution and optimization. At the system level, attribution has also been used to guide architecture and optimization decisions. Yang et al. (2025) evaluate functional components through Shapley values under backbone-model substitution. Xia et al. (2026) introduce CG-OPO to account for workflow dependencies, assigning zero utility to non-viable coalitions, and use removal-based attribution to select agents for prompt refinement. Taken together, however, existing approaches typically instantiate attribution with fixed design choices, making it difficult to isolate how these choices shape the resulting attribution. We address this gap with three contributions: we (i) unify previously separate design choices within a common attribution framework, (ii) systematically benchmark their effects across diverse MAS structures, and (iii) connect attribution to downstream system analysis and intervention.

3 A Unified Framework for Agent Attribution

In this work, we introduce a unified framework for agent attribution by formalizing it as a protocol-conditioned cooperative game (Covert et al., 2021). This view unifies existing approaches and makes explicit what question each attribution design answers.

3.1 Agent Attribution as a Cooperative Game

We formalize agent attribution through intervention games that quantify how changes to individual agents affect system-level behavior. This establishes the foundation of our work.

Definition 1. *A multi-agent system induces a baseline game represented by the tuple $(\mathcal{A}, \Gamma, \nu)$, where $\mathcal{A} = \{a_1, \ldots, a_n\}$ is the set of agents, Γ specifies the reference MAS architecture, including roles, routing, communication structure, and execution rules for configurations with absent agents, and $\nu : 2^{\mathcal{A}} \to \mathbb{R}$ is the characteristic utility function. Given a task distribution $\mathcal{D}$, the utility of an active coalition $C \subseteq \mathcal{A}$ is*

$$\nu(C) = \mathbb{E}_{x\sim\mathcal{D},\, \xi\sim\Xi}\left[\mathcal{G}\left(\tau^{\Gamma}_{C,x,\xi}\right)\right]. \tag{1}$$

where $\tau^{\Gamma}_{C,x,\xi}$ denotes the execution trace generated by the original agents in C, with all other agents absent, on input x under architecture Γ and randomness ξ. Here, ξ captures stochasticity from the LLM and/or environment dynamics, and $\mathcal{G}$ maps the execution trace to a scalar score, such as task success or token usage.

We generalize this baseline to intervention protocols by introducing protocol-specific reference states, denoted by $\mathbf{b} = (b_1, \ldots, b_n)$. For an intervention coalition $S \subseteq \mathcal{A}$, agents in S adopt their respective reference states, while all other agents retain their original implementations:

$$m_i^{\mathbf{b}}(S) = \begin{cases} b_i, & \text{if } a_i \in S, \\ a_i, & \text{if } a_i \notin S. \end{cases} \tag{2}$$

We define the corresponding expected utility as:

$$\nu_{\mathbf{b}}(S) = \mathbb{E}_{x\sim\mathcal{D},\xi\sim\Xi}\left[\mathcal{G}\left(\tau^{\Gamma}(\mathbf{m}^{\mathbf{b}}(S), x, \xi)\right)\right]. \tag{3}$$

For any $S \subseteq \mathcal{A}\backslash\{a_i\}$, the difference $\nu_{\mathbf{b}}(S\cup\{a_i\}) - \nu_{\mathbf{b}}(S)$ captures the marginal effect of intervening on a_i, holding the intervention status of all other agents fixed. We therefore define:

$$\phi_i^{\mathbf{b}} = \sum_{S\subseteq\mathcal{A}\backslash\{a_i\}} w_i(S)\left[\nu_{\mathbf{b}}(S\cup\{a_i\}) - \nu_{\mathbf{b}}(S)\right]. \tag{4}$$

Thus, $\phi_i^{\mathbf{b}}$ quantifies the effect of intervening on a_i, aggregated across intervention coalitions according to w_i. An attribution query is therefore specified by $(\mathcal{G}, w_i, \mathbf{b})$, where $\mathcal{G}$ specifies the target behavior, w_i determines the weighting over intervention coalitions, and $\mathbf{b}$ defines the intervention protocol.

3.2 Coalition Distributions for MAS

Rather than treating the coalition distribution $w_i(S)$ as universally fixed, our framework treats it as a design choice. The weights $w_i(S)$ specify which coalitions are used to estimate the contribution of agent a_i, and different choices correspond to different attribution methods and computational costs.

We consider a spectrum of coalition distributions. Leave-One-Out (LOO) measures the effect of intervening on a_i while all other agents remain unchanged, making it efficient but less sensitive to interactions that emerge when other agents are also intervened on. In contrast, combinatorial methods are more computationally expensive but evaluate marginal contributions across coalitions of varying sizes. The Shapley value (Shapley et al., 1953) averages over all possible coalitions, while Owen values (Owen et al., 1977) incorporate group structure by partitioning agents into modules. Myerson values (Myerson, 1977) can additionally impose graph-based constraints through connected components of an undirected attribution graph derived from the MAS communication structure. We compare these choices to examine how coalition distributions affect attribution across MAS architectures. Further details are provided in Section A.

3.3 Intervention Protocols in MAS

The reference vector $\mathbf{b}$ determines the intervention evaluated by attribution. We consider four protocols: agent ablation, introspective removal, model replacement, and prompt optimization. Under our convention, attribution measures the utility change from applying the intervention to a_i.

Agent ablation. Agent ablation sets $b_i = \varnothing$, removing a_i from the evaluated configuration. While this aligns with the notion of player removal in classical cooperative game theory, ablating indispensable

roles such as routers or orchestrators can result in system failures. A common workaround is to assign zero utility to non-executable coalitions (Xia et al., 2026), but it may conflate an agent's task contribution with its role in maintaining system executability.

Introspective removal. Cui et al. (2025) approximate ablation by prompting agents to revise their answers while disregarding another agent's responses. We denote simulated absence by $b_i = \varnothing_{\text{sim}}$, using this notation as shorthand for counterfactual utility estimation rather than a deployable implementation. The utility is estimated through in-context reasoning or an LLM judge instead of executing the ablated configuration. This approximates agent ablation, with fidelity depending on the evaluator and its prompts.

Model replacement. Prior work evaluates model replacement in workflows with four predefined components (Yang et al., 2025). We study this intervention across MAS with different agent roles and communication topologies. Specifically, we set $b_i = \tilde{a}_i$, where $\tilde{a}_i$ uses a different backbone model while retaining the agent's role, prompt, and tool access. Unlike ablation, replacement preserves the role and configured communication graph within each comparison. The resulting attribution also depends on the chosen replacement model, measuring the utility change from model substitution.

Prompt optimization. Prior work (Xia et al., 2026) uses removal-based Shapley attribution to select agents for prompt refinement. In contrast, we treat prompt optimization (Khattab et al., 2024; Agrawal et al., 2026) itself as an intervention when computing attributions. We set $b_i = a_i^{\text{opt}}$, where a_i^{opt} uses an optimized prompt $\tilde{p}_i$ while retaining the agent's role, backbone model, tool access, and configured communication links. With $\tilde{p}_i$ held fixed across coalition evaluations, the resulting attribution measures the utility change from that update.

3.4 Behavior Metrics

The final query component, $\mathcal{G}$, specifies what behavior is being attributed. Given an execution trace, it returns a scalar score, such as task success, token usage, or failure rate. The coalition utility $\nu_{\mathbf{b}}(S)$ is the expected value of this score over inputs and system randomness. Therefore, the choice of $\mathcal{G}$ determines how attribution scores should be interpreted: Accuracy metrics attribute contributions to task performance, while cost-based metrics attribute contributions to resource use.

Together, these components formalize MAS attribution in terms of the intervention protocol $\mathbf{b}$, the attribution distribution $w_i(S)$, and the behavior metric $\mathcal{G}$, providing a unified framework for analyzing MAS under different attribution queries.

4 Benchmarks and MAS Architectures

4.1 Benchmark Datasets

We instantiate our environments using the framework of Kim et al. (2025)[1] and evaluate on four benchmarks: (1) PlanCraft (Dagan et al., 2024), a Minecraft-based environment for assessing planning and tool use; (2) WorkBench (Styles et al., 2024), which evaluates planning and tool selection in business workflows; (3) BrowseComp-Plus (Chen et al., 2025b), which evaluates deep-research agents by retrieval, reasoning, and tool usage over human-verified documents; and (4) LiveCodeBench (Jain et al., 2025), which evaluates code generation on competitive-programming problems. Please refer to Section B for dataset details.

4.2 Communication Topologies

To study MAS behavior across these environments, we adopt four common collaboration topologies from Kim et al. (2025). Each topology is defined by a fixed set of agents $\mathcal{A}$ and directed communication links E (Figure 2).

Independent: $E_{\text{ind}} = \varnothing$, where agents operate independently without inter-agent communication. The system returns the first successful attempt, or the first completed attempt if none succeeds.

Decentralized: $E_{\text{decent}} = \{(a_i, a_j) : a_i, a_j \in \mathcal{A}, i \neq j\}$, a peer-to-peer network where agents work in parallel and share their findings with all peers between rounds.

[1] https://github.com/ybkim95/agent-scaling

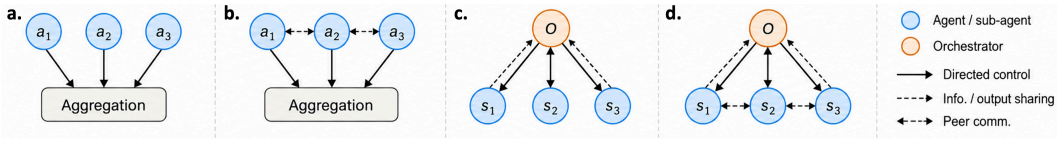


Figure 2: MAS communication graphs: **(a)** Independent **(b)** Decentralized **(c)** Centralized **(d)** Hybrid.

Centralized: $E_{\text{cent}} = \{(a_{\text{orch}}, a_i), (a_i, a_{\text{orch}}) : a_i \in \mathcal{A} \setminus \{a_{\text{orch}}\}\}$, where an orchestrator a_{orch} assigns subtasks, exchanges findings with each worker over multiple rounds, and synthesizes the final answer.

Hybrid: $E_{\text{hyb}} = E_{\text{cent}} \cup \{(a_i, a_j) : a_i, a_j \in \mathcal{A} \setminus \{a_{\text{orch}}\},\ i \neq j\}$, which augments the centralized topology with peer-to-peer links between all workers.

4.3 MAS Configurations

For each benchmark, we use a fixed set of role-specialized agents with distinct personas, task-specific prompts, and tool access. To keep exact computation of combinatorial attribution methods, such as Shapley values, feasible, we use five agents for PlanCraft and WorkBench and four subagents for LiveCodeBench and BrowseComp-Plus; centralized and hybrid topologies include one additional orchestrator. Following Kim et al. (2025), we report mean task success. For our primary experiments on four datasets, all agents use GPT-5.4 with medium reasoning, and communicating topologies run for at most five rounds. We additionally evaluate alternative backbones in Section E.7. Detailed agent roles, implementation details, and prompt templates are provided in Sections C and F.

4.4 Attribution Computation & Intervention Protocol Implementation

For agent ablation, we remove the target agent from the system. For introspective removal, an LLM judge estimates the effect of removing the target agent without modifying the deployed MAS, using the prompt in Section F.3. For model replacement, we substitute the target agent's backbone with GPT-OSS-120B (Agarwal et al., 2025), the strongest open-weight model from the same provider. For prompt optimization, we run GEPA (Agrawal et al., 2026) for five iterations with GPT-5.5 as the reflector, jointly optimizing all agent prompts on 100 examples disjoint from evaluation and split evenly between training and validation. Optimization uses the benchmark-specific evaluation objective, and each resulting agent prompt defines its intervention.

For each benchmark, we use 100 instances for attribution estimation and a disjoint set of 200 instances for evaluating top-k interventions, with agents ranked in descending order of their protocol-specific attribution. Each MAS configuration is evaluated over three independent runs.

5 Experiments & Results

We organize our experiments around practical questions and assumptions that arise in prior work. For each, we use our framework to isolate a relevant attribution design choice—intervention protocol, coalition distribution, or evaluation objective—and examine how it shapes agent importance across benchmarks and communication topologies.

5.1 Effect of Intervention Protocols

Fidelity of introspective removal. To avoid costly re-execution, prior work (Cui et al., 2025) derives attribution via counterfactual reasoning about MAS behavior when a targeted agent is removed. This approximation is reliable only if the inferred counterfactual utilities match those obtained by agent ablation. To ensure our analysis remains agnostic to the choice of coalition distribution, we directly compare introspective removal and agent ablation utilities across all $2^n - 1$ coalitions and compute R^2 and Spearman's ρ.

As shown in Figure 3, introspective removal generally overestimates coalition utility relative to agent ablation. At the instance level, introspective predictions disagree with actual execution on 30–52% of PlanCraft cases, with errors dominated by false positives: they often predict success when the instance actually fails. Moreover, increasing model capability does not consistently improve fidelity; GPT-5.4 does not outperform GPT-5-mini and exhibits stronger overestimation. This bias is reflected

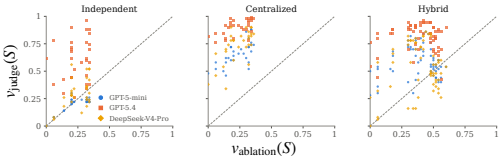


Figure 3: Introspective removal (v_{Judge}, y-axis) versus agent ablation coalition utility (v_{ablation}, x-axis) on PlanCraft.

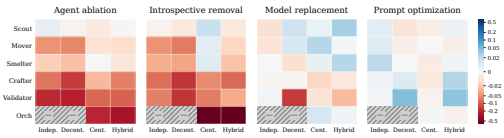


Figure 4: Agent Shapley values across intervention protocols on PlanCraft. Blue (red) indicates positive (negative) effects on task success.

Table 1: Agreement (R^2 / Spearman ρ) between introspective removal and agent-ablation coalition values on PlanCraft across all coalitions and backbone models, averaged over three runs.

Judge Model	Ind.	Decent.	Cent.	Hybrid
DeepSeek-V4-Flash	−1.05 / 0.47	−2.03 / 0.52	**-11.93 / 0.80**	**-0.99 / 0.14**
DeepSeek-V4-Pro	−0.77 / 0.43	−1.12 / 0.56	−21.46 / 0.69	−2.16 / 0.01
GPT-5-mini	**0.38 / 0.47**	**0.78 / 0.72**	−15.61 / 0.74	−2.16 / −0.08
GPT-5.4	−17.90 / 0.18	−10.21 / 0.56	−39.48 / 0.67	−7.79 / −0.06

Table 2: Normalized attribution entropy and token usage averaged across PlanCraft. Lower entropy indicates more concentrated attribution; shaded cells denote entropy < 0.6.

Intervention	Ind.	Decent.	Cent.	Hybrid	Tokens
Agent ablation	0.724	0.713	0.580	0.599	2.17M
Introspective removal	0.860	0.808	0.422	0.424	409.7K
Model replacement	0.711	0.469	0.929	0.817	1.51M
Prompt optimization	0.796	0.657	0.811	0.728	8.13M

in coalition-level agreement (Table 1). In the independent topology, the strongest judge (GPT-5-mini) achieves moderate agreement $R^2 = 0.38; \rho = 0.47, p = .008$, while all judge models obtain negative R^2 in centralized topologies despite moderate rank correlation. Agreement is even weaker on a more challenging BrowseComp-Plus (Table 3). We additionally test standard judge-enhancement strategies (Wang et al., 2022; Liu et al., 2023b; Madaan et al., 2023), but none consistently improves agreement (Table 4). Taken together, these results show that introspective removal, despite being substantially cheaper (Table 5), does not reliably approximate agent ablation, and that stronger judges or refinement strategies do not close this gap.

Distinct notions of agent importance. The disagreement between introspective removal and actual ablation suggests that agent importance is intervention-dependent rather than intrinsic. While prior work typically evaluates contribution under a single fixed intervention (Cui et al., 2025; Hua et al., 2026; Xia et al., 2026), our framework makes this choice explicit: each intervention induces a distinct attribution game (Figure 4,Section 3.3). To quantify this, we compare the concentration of attributions[2] across intervention games using normalized Shannon entropy (Shannon, 1948)[3].

As shown in Table 2, intervention protocols differ most clearly under hierarchical coordination on PlanCraft. In centralized and hybrid topologies, agent ablation and introspective removal yield concentrated attribution ($H_{\text{norm}} = 0.580/0.599$ and $0.422/0.424$, respectively), whereas model replacement ($0.929/0.817$) and prompt optimization ($0.811/0.728$) distribute attribution more broadly. This contrast reflects the intervention being measured: removal-based protocols emphasize agents whose absence disrupts the system, whereas replacement and prompt optimization measure the value of modifying agents while preserving their roles and communication structure. A broadly similar pattern holds under LOO, Owen, and Myerson (Tables 6 to 8). The protocols also differ in attribution cost, with prompt optimization requiring additional optimization and MAS execution, while introspective removal relies only on judge-based evaluation (Table 5).

Key Takeaway: Agent importance is not a single fixed quantity: importance depends on the intervention used to define it, leading to distinct attribution games and trade-offs.

5.2 Choice of Coalition Distribution

Rank agreement across coalition distributions. Existing work typically adopts a single coalition distribution, such as Shapley, without examining whether this choice changes which agents are identified as important (Cui et al., 2025; Ebrahimi et al., 2025; Yang et al., 2025; Xia et al., 2026). We compare LOO, Shapley, Owen, and Myerson (Section A) and measure rank agreement using Spearman correlation.

Across intervention protocols and communication topologies, the combinatorial methods generally yield highly consistent rankings despite differences in attribution magnitude, with pairwise $\rho =$

[2]Shapley values are shown here; results for other coalition distributions are reported in Tables 6 to 8.

[3]$H_{\text{norm}}(\phi) = -(\log n)^{-1} \sum_i p_i \log p_i$, where p_i is the normalized absolute attribution of agent i.

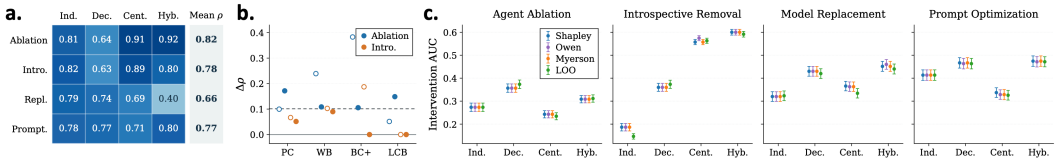


Figure 5: **(a)** LOO–Shapley Spearman correlation across intervention protocols and communication topologies, averaged over datasets. **(b)** Change in correlation by dataset. Hollow/solid markers denote centralized/hybrid topologies. **(c)** Rank-guided intervention AUC with 95% confidence intervals across coalition distributions and communication topologies on PlanCraft.

0.76–1.0 among Shapley, Owen, and Myerson, averaged over datasets and interventions (Figure 10). Notably, LOO also shows agreement with Shapley, with $\rho = 0.66$–0.82 across interventions, averaged over topologies (Figure 5(a)). To better understand this agreement, we further examine how it varies across intervention protocols and system structures. For agent ablation, we find more stable rankings when the system contains an essential agent, such as the Coder in LiveCodeBench or the orchestrator in centralized and hybrid systems, where coalitions without that agent have low or zero utility.

To quantify the effect of the essential agent, we recompute Shapley values with the essential agent always present.[4] Removing this essential-agent-induced weighting reduces LOO–Shapley correlation among the remaining agents by a mean of 0.113 (Figure 5(b)). Similarly, model replacement also weakens these structural utility gaps by preserving agent roles and yields lower rank agreement ($\rho = 0.66$; Figure 5(a)). A complementary case is agent redundancy: an agent may be redundant in the full team yet useful in smaller coalitions, causing LOO and Shapley to diverge. For example, on WorkBench, two agents share the same CRM–Email Generalist role (Section C.4), and adding the second provides no additional gain. LOO therefore assigns near-zero importance to most agents when evaluated against the full coalition, whereas Shapley captures their contributions across smaller coalitions, yielding much lower agreement ($\rho = 0.10$–0.50; Figure 10).

Intervention effectiveness. Finally, we evaluate whether ranking agreement translates into similar intervention outcomes by sequentially intervening on agents according to their attribution ranking. We summarize effectiveness using top-k intervention AUC with 95% confidence intervals (Petsiuk et al., 2018). Within each intervention protocol and communication topology, rank-guided intervention AUCs differ marginally across coalition distributions (Figure 5(c)). This holds not only for the independent topology but also for systems with communication graphs. For example, under decentralized agent ablation, top-k AUCs range from $0.263\ (0.026)$ to $0.270\ (0.033)$. We observe the same pattern on WorkBench, BrowseComp-Plus, and LiveCodeBench (Figure 12); paired ΔAUC comparisons across coalition distributions further confirm that these differences remain small (Figure 11).

Key takeaway: When the attribution game induces clear utility gaps—e.g., via intervention protocols or agent capability asymmetries—LOO can serve as an effective proxy for combinatorial attribution, recovering similar rankings at 2.2–11.9$\times$ lower token cost (Table 9).

5.3 Performance–Cost Trade-offs of Intervention Protocols

Prior work (Xia et al., 2026; Ebrahimi et al., 2025; Yang et al., 2025) uses attribution to select intervention targets but overlooks the resulting performance–cost trade-offs. This is particularly important in MASs, where inference costs can be substantial (Smit et al., 2023; Zeng et al., 2025). We therefore evaluate these trade-offs by ranking agents with LOO, applying the corresponding intervention to the selected k agents, and measuring task success and inference token usage.

Agent ablation and introspective removal. Our results show that removing agents with the highest ablation attribution can improve both performance and inference cost, but these gains diminish with deeper removal (Figure 6). For example, on WorkBench, ablation at $k = 1$ improves success by 3.5 percentage points while reducing token usage by 26.5% relative to the full system. However, on the more challenging BrowseComp-Plus benchmark, performance declines from $k = 1$. Overall, further removal reduces token usage but degrades performance significantly across datasets. For introspective removal, agents are removed based on judge-based attribution ranking. This yields similar token savings but less consistent performance than direct ablation.

[4]For essential agent e, we hold e un-intervened and recompute Shapley values over intervention coalitions $S \subseteq \mathcal{A} \setminus \{e\}$

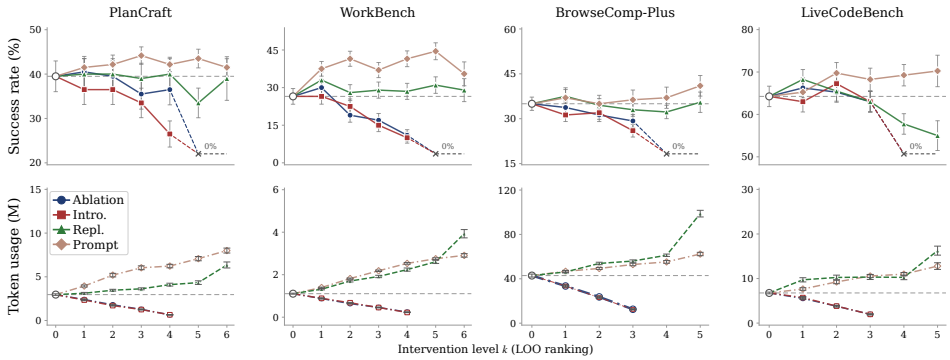


Figure 6: Task success (top) and inference cost (bottom) under top-k agent interventions across datasets. Error bars show 95% confidence intervals over communication topologies and three runs.

Model replacement. Model replacement preserves agent roles while changing their underlying models. With GPT-OSS-120B, replacing the $k = 1$–3 highest-ranked agents on WorkBench improves success by 1.5–6.5 percentage points over the baseline, while increasing total token usage by 19–73%. On BrowseComp-Plus, success improves by 2.5 percentage points at $k = 1$ but falls below baseline at $k = 4$, with token usage reaching $1.43\times$ the baseline. On LiveCodeBench, success similarly declines with replacement depth, from 68.25% at $k = 1$ to 57.75% at $k = 4$, while token usage reaches approximately $1.5\times$ the baseline. We further evaluate different replacement models and find that performance does not improve monotonically with substitute model size (Section E.8). Thus, while model replacement can improve performance at shallow depths, its effect depends on the chosen substitute, and deeper replacement may both degrade performance and increase token consumption.

Prompt optimization. Prompt optimization (Khattab et al., 2024; Agrawal et al., 2026) modifies agent behavior while preserving models and communication structure. On WorkBench, optimizing the top-k agents improves success by 10.5–15.0 percentage points for $k = 1$–3; on PlanCraft, the gain peaks at 5 points at $k = 3$; and on LiveCodeBench, optimizing the top two agents raises success from approximately 64% to 70%, while token usage increases from 7M to 9M (Figure 6). We find that optimizing more agents does not consistently improve performance. On LiveCodeBench, optimizing the top two agents achieves 69.8% success, comparable to 69.3% with four agents, while using 16% fewer tokens (9.2M vs. 11.0M). On PlanCraft, optimizing three agents outperforms optimizing all agents (44.2% vs. 41.5%) with 25% lower token usage (6.0M vs. 8.0M). Upon examining the optimizer prompts, it often replaces short descriptions with task-specific examples, procedures, and verification steps (Section F.4). Overall, our results show that targeted prompt optimization can match or outperform system-wide prompt optimization with substantially lower inference cost.

Key takeaway: Attribution-guided interventions offer the strongest performance–cost trade-offs at modest intervention depths. Deeper interventions provide diminishing returns: ablation increasingly sacrifices performance, while replacement and prompt optimization can increase inference cost without consistent performance gains.

5.4 Can removal-based attribution guide prompt optimization?

To guide agent selection for prompt optimization, Xia et al. (2026) propose CG-OPO, which computes Shapley values with agent ablation to identify bottleneck agents. However, under our framework, this creates an intervention–attribution mismatch: importance under removal need not reflect prompt-improvement potential. We therefore compare CG-OPO with top-k selection using LOO and Shapley attribution based on the prompt-optimization defined in our framework (Equation (4)), evaluating task success and attribution cost.

As shown in Figure 7(a), this mismatch matters in practice. Although ablation-based selection improves mean success, prompt-based attribution performs better at every intervention depth. At $k = 1$, prompt LOO and prompt Shapley achieve 48.0% and 49.1% success, respectively, compared with 44.2% for CG-OPO. The same pattern holds across benchmarks, with the largest gains on WorkBench (Figure 14). We also find that prompt LOO tracks prompt Shapley while using 79% fewer attribution tokens than prompt Shapley and 12% fewer than CG-OPO (Figure 7(b)).

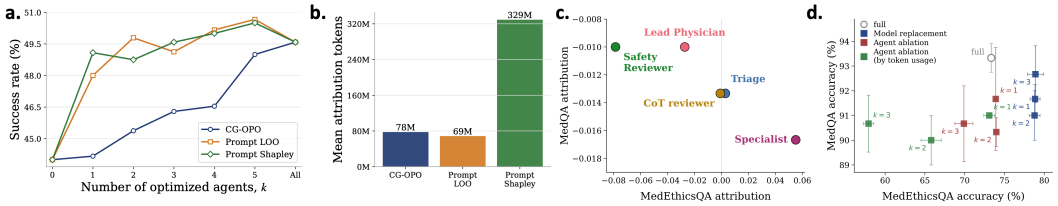


Figure 7: **(a)** Mean task success under top-k prompt optimization using CG-OPO, prompt-LOO, or prompt-Shapley. **(b)** Mean attribution token usage. Panels (a–b) are averaged across datasets and communication topologies. **(c)** Model-replacement attribution for MDTeamGPT on MedQA (y-axis) and MedEthicsQA (x-axis). **(d)** MDTeamGPT performance under top-k model replacement and agent ablation, with agents ranked by the corresponding MedEthicsQA attribution; green denotes a token-usage-based ablation baseline.

Key takeaway: Matching attribution to the intervention improves target selection: prompt LOO identifies more effective optimization targets at lower attribution cost than ablation-guided prompt optimization.

5.5 Evaluation Objectives Shape Agent Attribution: An Audit of MDTeamGPT

Prior work typically attributes agents with respect to a single evaluation metric (Yang et al., 2025; Xia et al., 2026), leaving open whether agent importance is stable across objectives. Our framework makes this dependence explicit: changing the evaluation metric changes the utility function and therefore induces a distinct attribution game (Section 3.4). This is particularly important in domains where systems must satisfy multiple behavioral objectives. In healthcare, for example, medical MASs must account for safety and ethical reasoning in addition to diagnostic performance (Kim et al., 2024; Tang et al., 2024; Ong et al., 2024). To examine whether agent importance changes across such objectives, we audit MDTeamGPT (Chen et al., 2025a), a medical consulting MAS using GPT-5-mini as its backbone model, under two evaluation metrics: diagnostic accuracy on MedQA (Jin et al., 2021) and ethical reasoning on MedEthicsQA (Wei et al., 2025) (Section D).

We compute LOO attribution via model replacement with Qwen 3.5 (27B) under both objectives and find sharply different contribution patterns across metrics. Specifically, we observe a near-complete reversal in agent attribution rankings between diagnostic accuracy and ethics alignment scores (Figure 7c). For example, replacing the Safety Reviewer decreases MedQA accuracy by 0.010 and the ethics score by 0.08, indicating that its original backbone benefits both objectives. Conversely, replacing the Specialist modestly decreases accuracy but yields the largest improvement in ethics ($+0.06$). These results show that backbone utility in MDTeamGPT is strongly metric-dependent: Agents whose original backbones improve diagnostic performance may reduce ethics alignment.

These attribution differences help identify agents whose original backbones degrade ethical performance while contributing little to diagnostic accuracy: **Specialist**, **Triage**, and **CoT Reviewer**. Replacing these three agents increases the ethics score by 4 percentage points while preserving MedQA accuracy (Figure 7d). In contrast, agent ablation does not improve ethics and decreases the ethics score by 4.6 percentage points at $k = 3$, with modest reductions in MedQA accuracy.

Key takeaway: Agent importance depends on the evaluation objective: auditing across behavioral metrics reveals trade-offs and enables interventions that improve desired behaviors.

6 Conclusion

In this work, we formalize MAS attribution as an intervention-conditioned cooperative game and show that agent importance depends on the intervention protocol, coalition distribution, and target metric. Different interventions therefore answer different attribution questions, while simple LOO often recovers similar rankings to more expensive combinatorial methods. For optimization, modest removal or model replacement often provides the strongest performance–cost trade-off. Overall, attribution is most useful when aligned with the intended intervention and evaluation objective—which agent to change, how to change it, and what system behavior to improve. Together, these results provide a principled basis for analyzing and improving complex multi-agent LLM systems.

AI USE STATEMENT

In this work, we used generative AI tools to assist with figure generation from experimental results, language and grammar editing, code generation for method implementation, experiment execution, and statistical analysis. All AI-assisted code was reviewed, verified, and tested for correctness by the authors. Separately, LLMs are also an explicit object of study within the proposed framework: the introspective-removal protocol uses an LLM judge to estimate coalition utility, as described in Section 3.3.

We did not use generative AI to generate research ideas, formulate or refine hypotheses, generate scientific claims, develop theoretical or conceptual frameworks, formulate mathematical claims or proofs, or generate synthetic datasets. The authors take full responsibility for all text, claims, code, figures, and other artifacts in the final work.

REPRODUCIBILITY STATEMENT

We release all code and environment configurations needed to reproduce the experiments in this work at `https://anonymous.4open.science/r/agent-eval-B850`. Section A details the computation of each coalition distribution; Section B describes dataset construction and sources; Section C specifies the agent configurations and system implementations; and Section D provides implementation details for MDTeamGPT.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *Advances in Neural Information Processing Systems*, 38, 2026.

Kai Chen, Xinfeng Li, Tianpei Yang, Hewei Wang, Wei Dong, and Yang Gao. Mdteamgpt: A self-evolving llm-based multi-agent framework for multi-disciplinary team medical consultation. *arXiv preprint arXiv:2503.13856*, 2025a.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, et al. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv preprint arXiv:2508.06600*, 2025b.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Ian Covert, Scott Lundberg, and Su-In Lee. Explaining by removing: A unified framework for model explanation. *Journal of Machine Learning Research*, 22(209):1–90, 2021.

Yue Cui, Liuyi Yao, Zitao Li, Yaliang Li, Bolin Ding, and Xiaofang Zhou. Efficient leave-one-out approximation in llm multi-agent debate based on introspection. *arXiv preprint arXiv:2505.22192*, 2025.

Gautier Dagan, Frank Keller, and Alex Lascarides. Plancraft: an evaluation dataset for planning with llm agents. *arXiv preprint arXiv:2412.21033*, 2024.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

Sana Ebrahimi, Mohsen Dehghankar, and Abolfazl Asudeh. An adversary-resistant multi-agent llm system via credibility scoring. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pp. 1676–1693, 2025.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 3(4):6, 2023.

Yun Hua, Haosheng Chen, Shiqin Wang, Wenhao Li, Xiangfeng Wang, and Jun Luo. Shapley-coop: Credit assignment for emergent cooperation in self-interested llm agents. *Advances in Neural Information Processing Systems*, 38:88675–88702, 2026.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines. 2024.

Yubin Kim, Chanwoo Park, Hyewon Jeong, Yik S Chan, Xuhai Xu, Daniel McDuff, Hyeonhoon Lee, Marzyeh Ghassemi, Cynthia Breazeal, and Hae W Park. Mdagents: An adaptive collaboration of llms for medical decision-making. *Advances in Neural Information Processing Systems*, 37: 79410–79452, 2024.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, et al. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023a.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: Nlg evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 2511–2522, 2023b.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*, 2023c.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Dover: Intervention-driven auto debugging for llm multi-agent systems. In *International Conference on Learning Representations*, volume 2026, pp. 28059–28087, 2026.

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

Roger B Myerson. Graphs and cooperation in games. *Mathematics of operations research*, 2(3): 225–229, 1977.

Jasmine Chiat Ling Ong, Shelley Yin-Hsi Chang, Wasswa William, Atul J Butte, Nigam H Shah, Lita Sui Tjien Chew, Nan Liu, Finale Doshi-Velez, Wei Lu, Julian Savulescu, et al. Medical ethics of large language models in medicine. *NEJM AI*, 1(7):AIra2400038, 2024.

Guillermo Owen, R Hein, and O Moeschlin. Essays in mathematical economics and game theory. *R. Henn and O. Moeschlin, New York*, 1977.

Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. *arXiv preprint arXiv:1806.07421*, 2018.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186, 2024.

Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

Lloyd S Shapley et al. A value for n-person games. 1953.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Andries Smit, Paul Duckworth, Nathan Grinsztajn, Thomas D Barrett, and Arnu Pretorius. Should we be going mad? a look at multi-agent debate strategies for llms. *arXiv preprint arXiv:2311.17371*, 2023.

Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. Workbench: a benchmark dataset for agents in a realistic workplace setting. *arXiv preprint arXiv:2405.00823*, 2024.

Xiangru Tang, Anni Zou, Zhuosheng Zhang, Ziming Li, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. Medagents: Large language models as collaborators for zero-shot medical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 599–621, 2024.

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

Jianhui Wei, Zijie Meng, Zikai Xiao, Tianxiang Hu, Yang Feng, Zhijie Zhou, Jian Wu, and Zuozhu Liu. Medethicsqa: A comprehensive question answering benchmark for medical ethics evaluation of llms. *arXiv preprint arXiv:2506.22808*, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*, 2024.

Yihan Xia, Taotao Wang, Shengli Zhang, Zhangyuhua Weng, Bin Cao, and Soung Chang Liew. Hivemind: Contribution-guided online prompt optimization of llm multi-agent systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 29767–29774, 2026.

Yingxuan Yang, Bo Huang, Siyuan Qi, Chao Feng, Haoyi Hu, Yuxuan Zhu, Jinbo Hu, Haoran Zhao, Ziyi He, Xiao Liu, et al. Who's the mvp? a game-theoretic evaluation benchmark for modular attribution in llm agents. *arXiv preprint arXiv:2502.00510*, 2025.

Yuting Zeng, Weizhe Huang, Lei Jiang, Tongxuan Liu, Xitai Jin, Chen Tianying Tiana, Jing Li, and Xiaohua Xu. S2-mad: Breaking the token barrier to enhance multi-agent debate efficiency. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 9393–9408, 2025.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*, 2025.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Daisy Zhe Wang, Zhenhailong Wang, Cheng Qian, Robert Tang, Heng Ji, et al. Multiagentbench: Evaluating the collaboration and competition of llm agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8580–8622, 2025.

APPENDIX

A ATTRIBUTION METHODS AND COALITION WEIGHTING

We provide formal definitions of the attribution methods evaluated in our experiments.

Leave-One-Out (LOO) LOO evaluates each agent by intervening on that agent while leaving all others unchanged. Under our formulation, this corresponds to placing all attribution weight on the empty intervention set, $S = \varnothing$. The LOO attribution for agent a_i under intervention $\mathbf{b}$ is

$$\phi^{\mathbf{b}}_{\mathrm{LOO},i} = \nu_{\mathbf{b}}(\{a_i\}) - \nu_{\mathbf{b}}(\varnothing). \tag{5}$$

Thus, LOO measures the marginal effect of intervening on a_i relative to the original system, with all other agents unchanged. Positive values indicate that the intervention improves utility, whereas negative values indicate that it degrades utility.

Shapley Value (Shapley et al., 1953) The Shapley value averages the effect of intervening on an agent across all intervention coalitions, weighting each intervention set according to its size. For an agent $a_i \in \mathcal{A}$ with $n = |\mathcal{A}|$, the Shapley attribution under intervention $\mathbf{b}$ is

$$\phi^{\mathbf{b}}_{\mathrm{Shapley},i} = \frac{1}{n} \sum_{S \subseteq \mathcal{A} \setminus \{a_i\}} \binom{n-1}{|S|}^{-1} \left[\nu_{\mathbf{b}}(S \cup \{a_i\}) - \nu_{\mathbf{b}}(S)\right]. \tag{6}$$

The weighting assigns equal total mass to each intervention-set size. Thus, unlike LOO, which evaluates the intervention only when all other agents remain unchanged, Shapley averages its effect across intervention contexts of all sizes.

Owen Value (Owen et al., 1977) The Owen value extends the Shapley value to settings with a predefined coalition structure. Let $\mathcal{G} = \{G_1, \ldots, G_m\}$ be a partition of agents into groups, and let $G(i)$ denote the group containing agent a_i. The Owen value averages the effect of intervening on a_i over both the order in which groups join the coalition and the order in which agents within $G(i)$ join. Let $U_T = \bigcup_{G \in T} G$ denote the agents contained in the preceding groups. The Owen attribution under intervention $\mathbf{b}$ is

$$\phi^{\mathbf{b}}_{\mathrm{Owen},i} = \sum_{T \subseteq \mathcal{G} \setminus \{G(i)\}} \sum_{S \subseteq G(i) \setminus \{a_i\}} w(T,S) \left[\nu_{\mathbf{b}}(U_T \cup S \cup \{a_i\}) - \nu_{\mathbf{b}}(U_T \cup S)\right], \tag{7}$$

where

$$w(T,S) = \frac{|T|!(m - |T| - 1)!}{m!} \frac{|S|!(|G(i)| - |S| - 1)!}{|G(i)|!}. \tag{8}$$

Thus, Owen attribution measures the effect of intervening on an agent while respecting a predefined grouping structure over agents.

Myerson Value (Myerson, 1977) The Myerson value extends the Shapley value to games with graph-structured communication constraints. Let $g = (\mathcal{A}, E_g)$ be an undirected graph over agents. We derive g from the MAS communication graph Γ by ignoring edge direction, connecting two agents whenever a communication link exists in either direction. This yields a star graph for the centralized topology and complete graphs for the decentralized and hybrid topologies. For the independent topology, which has no inter-agent communication, we use a complete graph so that Myerson reduces to Shapley.

For any coalition $S \subseteq \mathcal{A}$, let S/g denote the connected components induced by S in g. Because $\nu_{\mathbf{b}}(\varnothing)$ need not be zero, we subtract this baseline before summing component utilities to avoid counting it once per connected component. We define the graph-restricted intervention-conditioned utility as

$$\nu_{\mathbf{b},g}(S) = \sum_{C \in S/g} \left[\nu_{\mathbf{b}}(C) - \nu_{\mathbf{b}}(\varnothing)\right]. \tag{9}$$

This reduces to the standard graph restriction when $\nu_{\mathbf{b}}(\varnothing) = 0$. The Myerson attribution for agent a_i under intervention $\mathbf{b}$ is then the Shapley attribution of this graph-restricted game:

$$\phi^{\mathbf{b}}_{\mathrm{Myerson},i} = \frac{1}{n} \sum_{S \subseteq \mathcal{A} \setminus \{a_i\}} \binom{n-1}{|S|}^{-1} \left[\nu_{\mathbf{b},g}(S \cup \{a_i\}) - \nu_{\mathbf{b},g}(S)\right]. \tag{10}$$

B DATASETS

B.1 PLANCRAFT

PlanCraft [5] is a Minecraft-inspired planning benchmark for evaluating LLM agents on structured crafting tasks (Dagan et al., 2024). It provides text-only and multimodal interfaces based on the Minecraft crafting GUI, requiring agents to reason over inventories, recipes, intermediate dependencies, and actions. The benchmark contains 1,145 training, 570 validation, and 580 test examples, with tasks grouped into five difficulty levels and an additional set of intentionally unsolvable tasks. It also includes the Minecraft Wiki as an external knowledge source for tool use and retrieval-augmented reasoning. In our pipeline, we use its first 400 instances in the fixed dataset order: 100 for attribution estimation, 100 for prompt optimization, and a disjoint set of 200 for evaluating interventions. The environment exposes four tools, `search`, `move`, `smelt`, and `impossible`. Every agent can `search` recipes, while the action tools are assigned by role: the mover has `move`, the smelter `smelt`, the crafter both, and only the validator may additionally declare a target `impossible`.

B.2 WORKBENCH

WorkBench[6] is a workplace-task benchmark for evaluating agents in realistic business environments (Styles et al., 2024). It provides a sandbox with five databases and 26 tools covering analytics, calendar management, CRM, email, and project management. The benchmark contains 690 tasks from 69 human-curated templates, organized into six categories: Analytics, Calendar, CRM, Email, Project Management, and Multi-domain. WorkBench uses outcome-centric evaluation, where success is determined by the final database state. We draw a stratified sample of 400 tasks proportional to the domain composition of the benchmark and force-include the ten tasks that span all five domains: 100 tasks for attribution estimation (70 single-domain and 30 cross-domain), 100 for prompt optimization, and a disjoint set of 200 for evaluating interventions (134 single-domain and 66 cross-domain). Agents have access to all 26 native WorkBench tools and a `submit` action.

B.3 BROWSECOMP-PLUS

BrowseComp-Plus[7] is a deep-research benchmark for evaluating search-augmented agents under controlled retrieval conditions (Chen et al., 2025b). It replaces live web search with a fixed corpus, enabling reproducible evaluation of retrieval, evidence use, reasoning, and answer synthesis. The benchmark contains 830 queries and 100,195 documents, with human-verified supporting documents and challenging negatives for each query. In our pipeline, we draw a random subset of 400 queries with fixed seeds: 100 for attribution estimation, 100 for prompt optimization, and a disjoint set of 200 for evaluating interventions, while retaining the full document corpus. Agents search the corpus with a dense retriever (Qwen3-Embedding-4B) that returns the top-5 documents with snippets per query, can retrieve any document in full by ID, and submit their final answer with `done`. Agents use the `search_documents` and `retrieve_document` tools, signal completion with `done`, and are graded by `gpt-4o-mini` following the BrowseComp grading protocol.

B.4 LIVECODEBENCH

LiveCodeBench[8] is a continuously updated benchmark for evaluating code generation on competitive programming problems (Jain et al., 2025). It collects problems from LeetCode, AtCoder, and Codeforces and evaluates generated solutions against hidden test cases. We use 400 problems from the `code_generation_lite` releases v4–v6 (443 problems in total, contests from 2023-08 to 2025-04), sampled with fixed seeds: 200 problems from the v5 and v6 releases (100 for attribution estimation and 100 for prompt optimization) and a disjoint set of 200 from the remaining problems of v4–v6 for evaluating interventions. Agents share a single solution buffer and may run the problem's public example tests; a solution is graded on the hidden test cases with a 6-second timeout per test, and success is measured by pass@1.

[5] https://github.com/gautierdag/plancraft

[6] https://github.com/olly-styles/WorkBench

[7] https://github.com/texttron/BrowseComp-Plus

[8] https://github.com/LiveCodeBench/LiveCodeBench

C AGENT CONFIGURATION

We evaluate each specialist partition under four multi-agent architectures that vary in their communication structure. Across architectures, specialization is implemented with the same two mechanisms: agent-specific personas inserted into each worker's system prompt, and agent-specific tool whitelists applied during environment initialization.

C.1 ARCHITECTURES.

Each architecture uses five base workers for PlanCraft and WorkBench, and four base workers for BrowseComp-Plus and LiveCodeBench.

Independent. Workers run in parallel without inter-agent communication or an orchestrator. Each worker attempts the full task independently with at least three and at most five iterations. The final answer returns the first successful attempt, falling back to the first completed attempt.

Centralized. A single lead agent decomposes the task, assigns one subtask to each worker slot, sends per-round coordination messages, and synthesizes the final answer once its stopping criterion is met. Workers observe only their own conversation and the orchestrator's messages; there is no direct peer-to-peer channel. The system runs for at most five coordination rounds, with three worker iterations per round.

Decentralized. Workers act as peers without an orchestrator. They exchange findings after each round and terminate when a 0.7 consensus threshold is reached. The system runs for at most five rounds, with three worker iterations per round.

Hybrid. This architecture combines centralized orchestration with direct peer-to-peer communication among workers. Workers receive both orchestrator messages and teammates' findings each round. The round and timeout budgets match the centralized setting.

C.2 ORCHESTRATOR AGENT

The centralized and hybrid architectures instantiate a single lead agent alongside the worker slots. Unlike workers, the lead agent has no access to environment action tools; it does not call tools such as `move`, `search_documents`, or `calendar_create_event`. Its interface is plain LLM generation, used for planning, coordination, stopping, and synthesis.

At the start of each task, the lead agent produces a JSON `OrchestrationPlan` containing one {`agent_id`, `objective`, `focus`} subtask per worker slot. If parsing or validation fails, the system falls back to a generic per-slot plan. During execution, the lead agent reads each worker's accumulated findings and the team-wide context, then sends short coordination messages before the next worker iteration. After each round, it decides whether the team has sufficient evidence to answer or should continue, subject to the global round and timeout budgets. Once stopping is triggered or the budget is exhausted, the lead agent synthesizes the final answer from the full findings buffer.

The lead agent has full read access to worker findings. In the centralized architecture, workers receive only the lead agent's coordination messages and have no direct peer channel. In the hybrid architecture, the same lead agent is used, but workers also receive peer findings between rounds. The independent and decentralized architectures do not instantiate a lead agent.

C.3 WORKER AGENT: PLANCRAFT

PlanCraft exposes four action tools: `search`, `move`, `smelt`, and `impossible`. Because the action space is small and crafting actions are largely sequential, all worker slots retain the universally useful `search` tool, while the remaining action tools are distributed with deliberate overlap.

Agent 1: Recipe Scout. Tools: `search`. Retrieves recipes, ingredients, and intermediate steps, but cannot modify the inventory.

Agent 2: Inventory Mover. Tools: `search`, `move`. Places items in the crafting grid and can complete recipes requiring only inventory movement, but cannot smelt.

Agent 3: Smelter. Tools: `search`, `smelt`. Converts raw materials using the furnace, but cannot move items within the inventory.

Agent 4: Crafter. Tools: `search`, `move`, `smelt`. Has the full constructive action set and is responsible for completing feasible recipes.

Agent 5: Validator. Tools: `search`, `move`, `smelt`, `impossible`. Has the same constructive tools as the Crafter and is the only slot allowed to declare a task impossible.

C.4 Worker Agent: WorkBench

WorkBench provides 26 native tools across five domains: calendar, email, analytics, project management, and CRM. It also includes shared company-directory lookup tools and a `submit` terminator. Our specialist partition assigns three single-domain specialists and two CRM–email generalists, encouraging cross-domain coordination on multi-domain tasks. Tool whitelists are specified as `fnmatch` globs over registered tool names. Every worker has access to the company-directory tools and `submit`.

Agent 1: Calendar Coordinator. Tools: `calendar_*`, `company_directory_*`, `submit`. Handles scheduling, meetings, and availability constraints.

Agent 2: Project Manager. Tools: `project_management_*`, `company_directory_*`, `submit`. Creates, updates, searches, and closes project-management tasks.

Agent 3: Analytics Analyst. Tools: `analytics_*`, `company_directory_*`, `submit`. Handles visitor counts, traffic sources, session metrics, and plot generation.

Agents 4–5: CRM–Email Generalists. Tools: `crm_*`, `email_*`, `company_directory_*`, `submit`. These two identically scoped agents share the customer-lookup and email-communication interface. In centralized and hybrid settings, the orchestrator coordinates them to reduce duplicated actions; in independent and decentralized settings, they act as a redundant pair.

For centralized and hybrid WorkBench runs, the orchestrator must use the literal slot identifiers `agent_1`, . . . , `agent_5` in the subtask `agent_id` field. Semantic names such as "calendar" or "analyst" are not accepted, so role information is encoded only in the `focus` field.

C.5 Worker Agent: BrowseComp-Plus

BrowseComp-Plus exposes three tools: `search_documents`, `retrieve_document`, and `done`. Since partitioning this small tool set would leave some workers without essential actions or simply duplicate the same allocation, all four workers share the full tool set. Specialization is instead encoded through personas that decompose BrowseComp-style queries into complementary verification axes.

Agent 1: Candidate Enumerator. Tools: `search_documents`, `retrieve_document`, `done`. Generates a ranked shortlist of candidate entities using concrete searchable hooks such as years, awards, roles, or events. Prioritizes recall and reports document-backed candidates for verification.

Agent 2: Biographical Verifier. Tools: `search_documents`, `retrieve_document`, `done`. Confirms or rejects biographical facts for each candidate, including dates, locations, family background, education, and other identity constraints.

Agent 3: Relationship and Event Tracker. Tools: `search_documents`, `retrieve_document`, `done`. Verifies family relationships and time-bound events, such as spouses, children, dated incidents, and constraints tied to the target entity.

Agent 4: Answer Synthesizer. Tools: `search_documents`, `retrieve_document`, `done`. Integrates teammates' findings, issues targeted and disambiguating queries, and commits the final answer once all question constraints are supported.

The `done` terminator is available to all BrowseComp-Plus workers because any worker may identify a sufficiently supported answer. In centralized and hybrid settings, the orchestrator selects the

committed finding used as the final response; in independent and decentralized settings, the registered answer is determined by aggregation or consensus over worker outputs. The full agent prompt templates are provided in Section F.

C.6 WORKER AGENT: LIVECODEBENCH

LiveCodeBench provides four native tools over a single candidate solution: `write_solution` replaces the candidate program, `read_solution` returns it, `run_public_tests` executes it against the problem's public examples, and `submit` declares it final. Following Hong et al. (2023), our specialist partition assigns one role per stage of a plan–code–test–review workflow, with role-exclusive tools: only one agent can edit the solution and only one can execute it, so progress requires information to pass between roles. Tool whitelists are specified as exact tool names. Every worker has access to `read_solution` and `submit`.

Agent 1: Planner. Tools: `read_solution`, `submit`. Analyzes the problem, identifies the algorithm, complexity bounds, edge cases, and I/O format, provides the Coder with a plan, and reviews the current solution against that plan.

Agent 2: Coder. Tools: `write_solution`, `read_solution`, `submit`. Turns the plan into complete Python 3 code, preserves starter-code signatures, and fixes failures reported by the Tester. This is the only agent that can modify the solution.

Agent 3: Tester. Tools: `run_public_tests`, `read_solution`, `submit`. Runs the public examples against the current solution, diagnoses failures such as wrong outputs, exceptions, and timeouts, and reports the required changes. This is the only agent that can execute code, and it cannot edit it.

Agent 4: Reviewer. Tools: `read_solution`, `submit`. Inspects the current solution for logic errors, off-by-one mistakes, unhandled edge cases, and performance problems on large inputs, and reports concrete fixes. It can neither edit nor execute code.

In decentralized, centralized, and hybrid settings, all agents operate on one shared solution buffer per task; in the independent setting, each agent maintains its own buffer. A `submit` call is an agent's vote that the current candidate is final and ends only that agent's turn for the current round, so the orchestrator may dispatch the same agent again in later rounds. The final candidate is graded against the hidden test cases.

For centralized and hybrid LiveCodeBench runs, the orchestrator must use the literal slot identifiers `agent_1`, ..., `agent_4` in the subtask `agent_id` field. Semantic names such as "planner" or "coder" are not accepted, so role information is encoded only in the `focus` field.

C.7 IMPLEMENTATION DETAILS

We implement our experiments using the agent-scaling framework[9] in Python 3.11, with LiteLLM[10] version 1.73.6 and LangChain version 0.3.26 [11]. All tasks are executed in a standard Python 3.11 environment. The BrowseComp-Plus, PlanCraft, WorkBench, and LiveCodeBench datasets are run directly, without sandboxing. We summarize the experimental configuration below.

Model Configuration. All experiments use the backbone models described in Section 4.3. LLM requests are routed through the OpenRouter [12] API using LiteLLM. We use a temperature of 0.0, corresponding to greedy decoding for pass@1 evaluation. We do not override top-p sampling, and we use the provider's default server-side maximum token limit.

Framework Hyperparameters. We retain the default hyperparameters of the framework. The outer coordination loop is limited to 5 rounds. Each worker agent is assigned at most 5 action iterations, with the minimum number of iterations set to 3. The memory of collected findings is capped at 100 entries. Each task has a wall-clock time limit of 600 seconds, and each worker has an individual timeout of 300 seconds.

[9] https://github.com/ybkim95/agent-scaling

[10] https://github.com/BerriAI/litellm

[11] https://github.com/langchain-ai/langchain

[12] https://openrouter.ai/

Cost Tracking. We record token usage and monetary cost using LiteLLM's built-in callbacks. These callbacks capture the exact prompt-token and completion-token counts returned by the OpenRouter API for every model call.

D Additional Details for MDTeamGPT

MDTeamGPT (Chen et al., 2025a) is configured as a multi-agent clinical consultation framework consisting of specialist doctor agents and auxiliary reviewer agents. The system follows a multi-round consultation workflow based on residual discussion and consensus aggregation. In this workflow, a Primary Care Doctor first triages the patient case and assigns it to appropriate specialist agents. The specialists then discuss the case over multiple rounds, incorporating prior-round information through a Historical Shared Pool. A Lead Physician summarizes and organizes the discussion, while reviewer agents verify safety and archive consultation experience for future use.

Primary Care Doctor. The Primary Care Doctor serves as the triage agent. Given the patient's symptoms, signs, and medical history, this agent assigns the case to appropriate specialist doctors for further consultation.

Specialist Doctors. Specialist doctors include roles such as General Internal Medicine Doctor, Pediatrician, and Radiologist. Each specialist provides diagnoses and treatment recommendations based on the patient case. During multi-round discussions, specialists refer to the Historical Shared Pool to incorporate insights from previous rounds before selecting an optimal treatment plan.

Lead Physician. The Lead Physician organizes and summarizes the diagnoses and treatment recommendations generated by the specialist agents. It categorizes specialist responses into four relation types: Consistency, Conflict, Independence, and Integration. These structured summaries are stored in the Historical Shared Pool, helping maintain an organized discussion process and reducing cognitive load across rounds.

Chain-of-Thought Reviewer. The Chain-of-Thought Reviewer extracts structured reasoning traces from each doctor's diagnostic process. Depending on whether the consultation outcome is correct or incorrect, the extracted statements are archived into either the Correct Answer Knowledge Base, denoted as CorrectKB, or the Chain-of-Thought Knowledge Base, denoted as ChainKB.

Safety and Ethics Reviewer. The Safety and Ethics Reviewer verifies and refines the final consultation outcome. It filters harmful, unsafe, or unethical suggestions to ensure that the final diagnosis and treatment recommendation satisfy safety and ethical constraints.

D.1 Datasets: MedQA and MedEthicsQA

We evaluate MDTeamGPT on two medical reasoning benchmarks: MedQA[13](Jin et al., 2021), using 100 sampled USMLE-style multiple-choice questions with four or five answer options, and MedEthicsQA[14](Wei et al., 2025), using 100 multiple-choice questions and 100 open-ended questions. MedQA assesses medical knowledge and practical clinical reasoning, while MedEthicsQA evaluates the system's ability to reason about medical ethics in both answer-selection and free-form response settings. For multiple-choice questions in both datasets, we report accuracy. For MedEthicsQA open-ended questions, we evaluate the final response using an LLM-as-a-judge protocol that compares the generated answer against the reference answer and assigns a binary correctness label; we report the resulting judge-based accuracy using the same formula.

[13] https://github.com/jind11/MedQA

[14] https://github.com/JianhuiWei7/MedEthicsQA

D.2 Implementation Details

Our implementation settings follow Chen et al. (2025a)[15] and are summarized below.

Experimental Environment. We implement the system in Python 3.13 using LangGraph for multi-agent orchestration. We implement vector similarity search for knowledge base retrieval with FAISS.

Backbone Model Configuration. In our experiments, MDTeamGPT uses GPT-5-mini as the backbone model for all agent configurations. For embedding textual content into the vector spaces of the knowledge bases, we use OpenAI's `text-embedding-3-small` model.

Framework Hyperparameters. The system conducts multi-round specialist discussions until all specialist doctor agents reach consensus, subject to a maximum of 6 rounds. If consensus is not reached within this limit, the final decision is determined by majority vote. For retrieval-based prompt enhancement from the knowledge bases, the system selects the top $K = 5$ most similar cases.

Temperature Settings. Specialist doctor agents use a temperature of 0.7 to encourage diverse clinical perspectives. The Lead Physician, Safety and Ethics Reviewer, and CoT Reviewer all use a temperature of 0.0 for deterministic outputs.

Knowledge Base Configuration. MDTeamGPT maintains two experience repositories: the Correct Answer Knowledge Base, CorrectKB, and the Chain-of-Thought Knowledge Base, ChainKB. In the main experiments, the framework stores consultation experience from 600 consultation rounds.

[15] `https://github.com/KaiChenNJ/MDTeamGPT`

Table 3: Agreement (R^2 / Spearman ρ) between introspective and agent-ablation coalition values across four benchmarks, averaged over 3 runs. Best results are shown in bold. ns denotes $p \geq 0.05$ for Spearman ρ.

Judge	Independent	Decentralized	Centralized	Hybrid
PlanCraft				
DeepSeek-V4-Flash	−1.05 / 0.47	−2.03 / 0.52	−11.93 / 0.80	−0.99 / 0.14
DeepSeek-V4-Pro	−0.77 / 0.43	−1.12 / 0.56	−21.46 / 0.69	−2.16 / 0.01
GPT-5-mini	0.38 / 0.47	0.78 / 0.72	−15.61 / 0.74	−2.16 / −0.08
GPT-5.4	−17.90 / 0.18	−10.21 / 0.56	−39.48 / 0.67	−7.79 / −0.06
WorkBench				
DeepSeek-V4-Flash	−5.84 / 0.92	−10.07 / 0.59	−5.26 / 0.85	−3.58 / 0.63
DeepSeek-V4-Pro	−4.41 / 0.92	−6.16 / 0.64	−2.20 / 0.71	−1.71 / 0.59
GPT-5-mini	−2.54 / 0.93	−1.10 / 0.84	−0.38 / 0.81	−0.32 / 0.61
GPT-5.4	−3.24 / 0.93	−11.30 / 0.63	−12.09 / 0.76	−11.21 / 0.66
BrowseComp-Plus				
DeepSeek-V4-Flash	−1532 / −0.73	0.27 / 0.71	0.08 / 0.92	−4.81 / 0.18^{ns}
DeepSeek-V4-Pro	−1552 / −0.72	0.31 / 0.75	0.69 / 0.81	−2.58 / 0.23^{ns}
GPT-5-mini	−2169 / −0.74	0.22 / 0.76	−1.20 / 0.36^{ns}	−1.04 / -0.27^{ns}
GPT-5.4	−2551 / −0.77	−0.42 / 0.72	−0.82 / 0.78	−0.14 / 0.13^{ns}
LiveCodeBench				
DeepSeek-V4-Flash	0.89 / 0.87	−0.02 / 0.83	−0.50 / 0.71	−0.08 / 0.62
DeepSeek-V4-Pro	0.96 / 0.80	0.50 / 0.82	−0.39 / 0.82	0.25 / 0.83
GPT-5-mini	0.96 / 0.76	0.30 / 0.77	−0.79 / 0.84	−0.26 / 0.80
GPT-5.4	0.79 / 0.82	−0.31 / 0.77	−2.26 / 0.64	−1.36 / 0.54

E Additional Results

E.1 Fidelity of Introspective Removal

Figure 8 compares judge-estimated and agent-ablation coalition utilities, while Table 3 reports the corresponding R^2 and Spearman ρ. Across benchmarks, judges often preserve the coarse ordering of coalitions while systematically shifting or dispersing their estimated utilities. This explains why moderate or high ρ can coexist with strongly negative R^2. Fidelity also depends on the spread of true coalition utilities: agreement is weakest when utilities are tightly concentrated and strongest when coalitions form well-separated groups.

PlanCraft. Rank agreement varies substantially by topology and judge. Decentralized and centralized systems retain moderate-to-high ρ, while centralized and hybrid systems have uniformly negative R^2, consistent with the large upward shifts in Figure 8. Thus, coalition ordering can remain partially preserved even when absolute utilities are poorly calibrated.

WorkBench. Rank agreement is strongest in the independent topology and remains moderate elsewhere ($\rho = 0.59$–0.85), yet all R^2 values are negative. The dominant error is therefore a systematic shift in coalition utility rather than a complete loss of rank information.

BrowseComp-Plus. Agreement varies strongly by topology. In independent systems, ablation utilities are uniformly low while judges assign higher utility 0.2–0.5, yielding negative rank correlation for all judges ($\rho = -0.77$ to -0.72). Agreement improves with communication: decentralized systems reach $\rho = 0.71$–0.76 with the best absolute agreement ($R^2 = -0.7$ to 0.3), and centralized systems reach $\rho = 0.78$–0.92 for all but GPT-5-mini, despite negative R^2. Hybrid systems remain weak ($\rho = -0.27$ to 0.23).

LiveCodeBench. Half of the coalitions have zero ablation utility because coalitions without the Coder cannot solve the task. Independent judges largely recover this structure: Coder-less coalitions receive utilities of 0.00–0.10, while Coder-containing coalitions lie at 0.6–0.8, yielding the strongest absolute agreement ($R^2 = 0.79$–0.96). Under communication, this separation weakens as Coder-less coalitions receive substantially higher estimates. Rank agreement therefore remains relatively strong ($\rho = 0.54$–0.84), while R^2 becomes negative because many zero-utility coalitions are overestimated.

Overall, judge capability does not monotonically improve either absolute or rank agreement. Stronger judges therefore do not reliably reproduce the coalition utilities obtained by actual ablation.

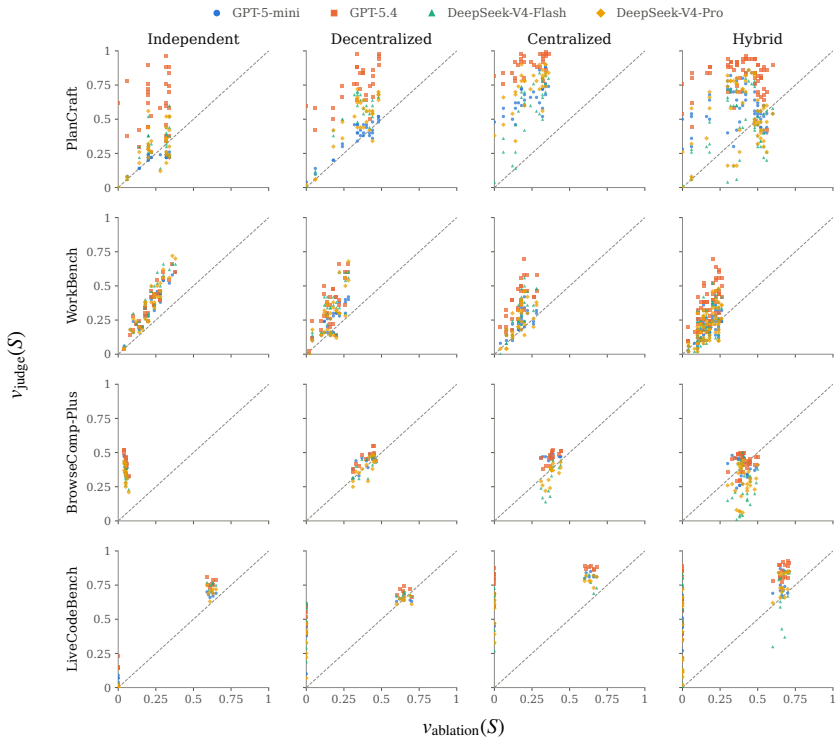


Figure 8: Judge-estimated versus actual coalition values across datasets and collaboration patterns. Each point denotes a coalition, with agent-ablation utility on the x-axis and introspective-removal utility on the y-axis; the dashed line indicates perfect agreement.

Table 4: Sensitivity of introspective removal to judge design (GPT-5-mini) on PlanCraft. Each cell reports agreement (R^2 / Spearman ρ) against agent-ablation coalition values, averaged over three runs. Best results within each topology are bolded.

Judge variant	**Indep.**	**Decent.**	**Cen.**	**Hybrid**
Baseline	0.38 / 0.47	0.78 / 0.72	−15.61 / 0.74	−2.16 / −0.08
G-Eval w counterfactual rubrics (Liu et al., 2023b)	−1.48 / 0.55	−2.28 / 0.46	−52.03 / 0.61	−33.03 / 0.63
Self-refinement (Madaan et al., 2023)	−0.57 / 0.54	−1.15 / 0.46	−43.58 / 0.61	−29.19 / 0.63
Self-consistency Wang et al. (2022)	0.38 / 0.66	0.34 / 0.57	−22.17 / 0.77	−17.12 / 0.65

E.1.1 Effectiveness of Judge Variants

In this section, we select the strongest introspective judge (Section 4.4)(GPT-5-mini) as the backbone and implement three variants: *Rubric-based Chain-of-Thought (CoT) prompting*(Liu et al., 2023b), augmented with an explicit counterfactual rubric; *Self-Consistency*(Wang et al., 2022), which majority-votes $K{=}3$ perturbed baseline judgments; and *Self-Refinement*(Madaan et al., 2023), which asks the CoT judge to critique and revise its initial prediction. All variants are evaluated on PlanCraft using the same coalitions, topologies, and three runs as exact removal.

Table 4 shows that none of these variants consistently improves both rank and absolute agreement across topologies. Self-consistency improves rank correlation in three of four topologies, but does not improve calibration and incurs $3\times$ the judge cost. Rubric-based CoT, self-consistency, and self-refinement perform worse, reducing rank agreement and increasing upward bias in estimated coalition utility. On Centralized and Hybrid, they overestimate mean coalition value by 0.43–0.47, despite true means of only 0.11–0.15, further degrading the already negative identity-line R^2. For reference, independent exact-removal re-runs achieve $\rho = 0.76$–0.86. These results suggest that standard judge-enhancement strategies do not resolve the limitations of introspective removal: additional reasoning or ensembling does not substitute for re-executing the coalition.

Table 5: Normalized Shapley attribution entropy ($H/\log_2 n$) across MAS architectures, with total evaluation token usage and cost (USD) over all $2^n - 1$ coalitions, macro-averaged over architectures. Shaded cells indicate high influence concentration (entropy < 0.6).

Intervention Protocol	Independent.	Decentralized.	Centralized.	Hybrid	Tokens	Cost ($)
PlanCraft						
Agent ablation	0.724	0.713	0.580	0.599	95.69M	264.47
Introspective removal	0.860	0.808	0.422	0.424	20.52M	6.70
Model replacement	0.711	0.469	0.929	0.817	80.97M	228.36
Prompt optimization	0.796	0.657	0.811	0.728	366.65M	708.51
WorkBench						
Agent ablation	0.976	0.865	0.739	0.670	21.65M	41.20
Introspective removal	0.960	0.829	0.720	0.579	9.23M	3.70
Model replacement	0.815	0.962	0.933	0.939	32.76M	54.61
Prompt optimization	0.803	0.948	0.755	0.822	108.22M	142.21
BrowseComp-Plus						
Agent ablation	0.948	0.986	0.629	0.581	537.88M	1417.20
Introspective removal	0.986	0.996	0.155	0.132	17.99M	5.49
Model replacement	0.928	0.912	0.827	0.895	586.60M	1500.49
Prompt optimization	–	0.934	0.857	0.698	826.53M	1502.61
LiveCodeBench						
Agent ablation	0.107	0.148	0.481	0.483	118.47M	436.97
Introspective removal	0.217	0.816	0.455	0.499	17.48M	5.80
Model replacement	0.315	0.632	0.754	0.662	109.50M	344.79
Prompt optimization	0.891	0.918	0.754	0.802	191.72M	538.57

Table 6: Normalized LOO attribution entropy ($H/\log_2 n$) across MAS architectures, with total evaluation token usage and cost (USD) over the n leave-one-out coalitions plus the full team, macro-averaged over architectures. Shaded cells indicate high influence concentration (entropy < 0.6).

Intervention Protocol	Independent.	Decentralized.	Centralized.	Hybrid	Tokens	Cost ($)
PlanCraft						
Agent ablation	0.457	0.815	0.534	0.632	21.48M	57.18
Introspective removal	0.717	0.906	0.751	0.513	2.69M	0.81
Model replacement	0.590	0.623	0.387	0.887	16.82M	48.14
Prompt optimization	0.657	0.772	0.947	0.925	71.38M	111.86
WorkBench						
Agent ablation	0.943	0.906	0.794	0.459	5.81M	11.34
Introspective removal	0.714	0.611	0.732	0.809	1.24M	0.51
Model replacement	0.793	0.826	0.678	0.974	6.59M	12.14
Prompt optimization	0.870	0.980	0.933	0.910	18.82M	19.76
BrowseComp-Plus						
Agent ablation	0.594	0.979	0.518	0.429	195.78M	512.70
Introspective removal	0.778	0.896	0.308	0.221	4.18M	1.17
Model replacement	0.792	0.765	0.984	0.960	199.67M	509.64
Prompt optimization	–	0.689	0.814	0.845	196.28M	339.80
LiveCodeBench						
Agent ablation	0.324	0.348	0.524	0.495	45.30M	168.95
Introspective removal	0.251	0.499	0.360	0.306	4.02M	1.29
Model replacement	0.721	0.901	0.815	0.716	36.76M	119.04
Prompt optimization	0.921	0.921	0.772	0.969	48.44M	125.65

Table 7: Normalized Myerson-value attribution entropy ($H/\log_2 n$) across MAS architectures, with total evaluation token usage and cost (USD) over the coalitions connected in the communication graph (star with the orchestrator as hub for centralized, complete otherwise), macro-averaged over architectures. Shaded cells indicate high influence concentration (entropy < 0.6).

Intervention Protocol	Independent.	Decentralized.	Centralized.	Hybrid	Tokens	Cost ($)
PlanCraft						
Agent ablation	0.724	0.713	0.580	0.599	95.69M	264.47
Introspective removal	0.860	0.808	0.422	0.424	17.63M	5.76
Model replacement	0.711	0.469	0.805	0.817	72.31M	202.87
Prompt optimization	0.796	0.657	0.773	0.728	328.49M	613.12
WorkBench						
Agent ablation	0.976	0.865	0.739	0.670	21.65M	41.20
Introspective removal	0.960	0.829	0.720	0.579	7.84M	3.19
Model replacement	0.815	0.962	0.900	0.939	28.93M	48.19
Prompt optimization	0.803	0.948	0.867	0.822	94.73M	122.32
BrowseComp-Plus						
Agent ablation	0.948	0.986	0.629	0.581	537.88M	1417.20
Introspective removal	0.986	0.996	0.155	0.132	15.69M	4.83
Model replacement	0.928	0.912	0.984	0.895	492.86M	1259.85
Prompt optimization	–	0.934	0.820	0.698	721.78M	1249.53
LiveCodeBench						
Agent ablation	0.107	0.148	0.481	0.483	118.47M	436.97
Introspective removal	0.217	0.816	0.455	0.499	15.31M	5.08
Model replacement	0.315	0.632	0.847	0.662	96.75M	304.49
Prompt optimization	0.891	0.918	0.655	0.802	170.63M	476.00

Table 8: Normalized Owen-value attribution entropy ($H/\log_2 n$) across MAS architectures, with total evaluation token usage and cost (USD) over all $2^n - 1$ coalitions, macro-averaged over architectures. The coalition structure is $\{\texttt{workers}\}, \{\texttt{orchestrator}\}$; without an orchestrator the Owen value equals the Shapley value. Shaded cells indicate high influence concentration (entropy < 0.6).

Intervention Protocol	Independent.	Decentralized.	Centralized.	Hybrid	Tokens	Cost ($)
PlanCraft						
Agent ablation	0.724	0.713	0.696	0.705	95.69M	264.47
Introspective removal	0.860	0.808	0.498	0.620	20.52M	6.70
Model replacement	0.711	0.469	0.952	0.908	80.97M	228.36
Prompt optimization	0.796	0.657	0.712	0.632	366.65M	708.51
WorkBench						
Agent ablation	0.976	0.865	0.811	0.828	21.65M	41.20
Introspective removal	0.960	0.829	0.785	0.752	9.23M	3.70
Model replacement	0.815	0.962	0.866	0.949	32.76M	54.61
Prompt optimization	0.803	0.948	0.687	0.806	108.22M	142.21
BrowseComp-Plus						
Agent ablation	0.948	0.986	0.860	0.859	537.88M	1417.20
Introspective removal	0.986	0.996	0.106	0.134	17.99M	5.49
Model replacement	0.928	0.912	0.956	0.902	586.60M	1500.49
Prompt optimization	–	0.934	0.669	0.560	826.53M	1502.61
LiveCodeBench						
Agent ablation	0.107	0.148	0.479	0.470	118.47M	436.97
Introspective removal	0.217	0.816	0.668	0.680	17.48M	5.80
Model replacement	0.315	0.632	0.836	0.692	109.50M	344.79
Prompt optimization	0.891	0.918	0.715	0.702	191.72M	538.57

E.2 AGENT-LEVEL EFFECTS OF EACH INTERVENTION.

Figure 9 reports the Shapley value of intervening on each agent. The intervention protocols differ substantially in both the magnitude and sign of their effects.

Removal. Agent ablation is predominantly harmful: removing an agent reduces success in 91% of cells. Attribution is also concentrated in structurally important roles. The orchestrator has the most negative ablation attribution in every centralized and hybrid system (-0.16 to -0.34), while the LiveCodeBench Coder dominates among workers, with ablation values of -0.62 and -0.64 in the independent and decentralized topologies. Similarly, PlanCraft assigns most worker credit to the Validator and Crafter, which have access to the full action set. Only 7 of 80 ablation cells are positive, and these gains are small (≤ 0.017). Introspective removal broadly preserves this structure but produces larger-magnitude values overall (median absolute value 0.048 vs. 0.026 for ablation), particularly for orchestrators.

Model replacement. Replacement produces substantially smaller effects (median absolute value 0.012) and more mixed signs, with 46% of cells positive. Most agents can therefore be replaced by GPT-OSS-120B with limited change in task success. Larger negative effects remain concentrated in capability-critical roles: replacing the LiveCodeBench Coder reduces success across all topologies, while replacing the PlanCraft Validator is also harmful in several settings. In contrast, replacing the orchestrator generally has little effect (-0.04 to $+0.03$).

Prompt optimization. Prompt optimization is the only intervention with predominantly positive effects, with 61% of cells improving task success. Gains are concentrated in particular agents, including the WorkBench Analytics and Calendar roles, while optimizing some agents decreases performance. Notably, the LiveCodeBench Coder—which dominates under ablation and replacement—gains only 0.02–0.03 from prompt optimization. Thus, agents that are critical to current system performance are not necessarily those with the greatest optimization headroom.

The identity of high-impact agents depends on the intervention: removal emphasizes structural necessity, replacement reflects dependence on backbone capability, and prompt optimization captures room for behavioral improvement.

E.3 RANK AGREEMENT ACROSS COALITION DISTRIBUTION

Figure 10 compares the rankings produced by LOO, Myerson, and Owen against Shapley across datasets, intervention protocols, and communication topologies.

LOO. LOO shows substantial agreement with Shapley, with mean correlations of 0.82, 0.70, 0.63, and 0.77 for ablation, introspective removal, replacement, and prompt optimization, respectively. Agreement is especially high for removal-based interventions in centralized and hybrid systems ($\rho = 0.78$–1.00), consistent with these settings emphasizing the full-coalition context captured by LOO. Model replacement yields the lowest agreement overall.

Myerson. Myerson coincides with Shapley whenever the communication graph is complete and also under centralized removal interventions. Differences therefore arise mainly for centralized replacement and prompt optimization, where correlations vary substantially across datasets.

Owen. Owen generally agrees more closely with Shapley than LOO, with mean correlations of 0.91, 0.89, 0.79, and 0.84 across the four intervention protocols. It achieves higher mean agreement than LOO in 12 of 16 dataset–protocol pairs.

Despite differences in attribution values, the resulting agent rankings are generally stable across coalition distributions, with the strongest agreement among the combinatorial methods and substantial agreement from the cheaper LOO approximation.

E.4 RANK-GUIDED INTERVENTION UTILITY ON WORKBENCH, BROWSECOMP-PLUS, AND LIVECODEBENCH.

Figure 12 extends the analysis in Figure 5(c) to the remaining benchmarks with GPT-5.4 teams. Agents are ranked by Shapley, Owen, Myerson, or LOO, and we evaluate the resulting top-k interventions using AUC over $k = 0, \ldots, N$.

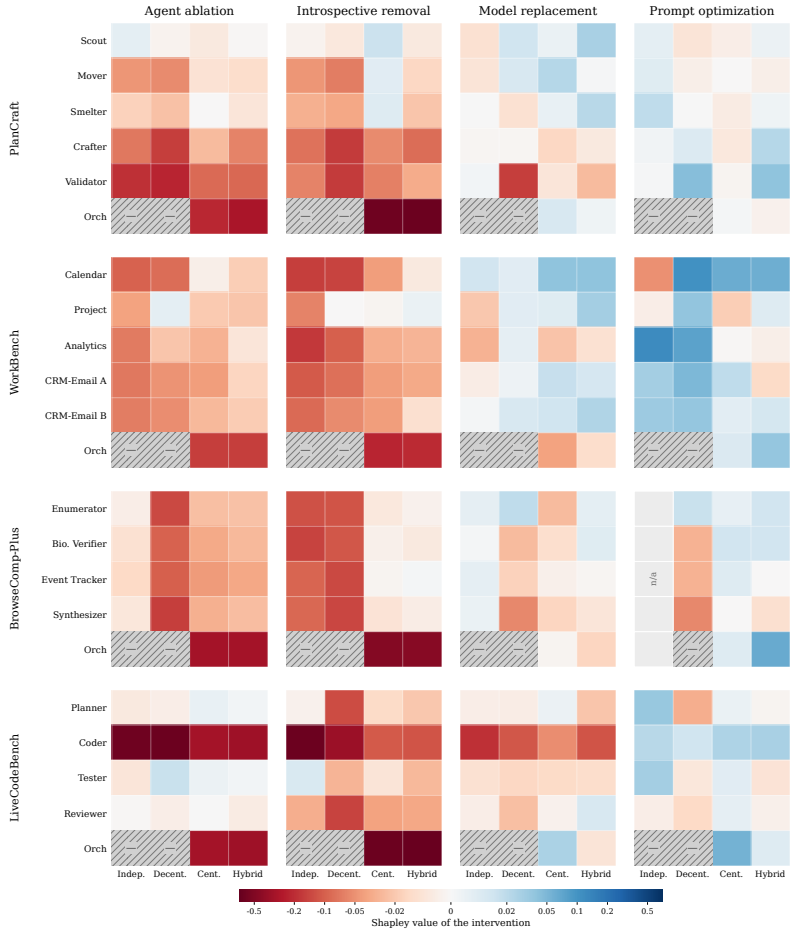


Figure 9: **Agent-level Shapley values across intervention protocols for GPT-5.4 teams.** Each cell shows the Shapley value of intervening on an agent (rows) under a communication topology (columns). Interventions are agent ablation, introspective removal, model replacement with GPT-OSS-120B, and prompt optimization. Blue (red) indicates a positive (negative) contribution to task success, using a shared symmetric logarithmic scale. Hatched cells denote agents absent from a topology, and n/a denotes unavailable prompt-optimization settings.

The four ranking methods yield nearly the same downstream intervention utility. To compare methods, we compute paired AUC differences from Shapley within each benchmark–protocol–topology setting (Figure 11). LOO trails Shapley by only 0.004 AUC on average (95% CI $[0.001, 0.006]$; Wilcoxon signed-rank $p = 0.008$), a small difference relative to a typical AUC of 0.38, and the maximum per-setting difference is 0.040. Owen and Myerson differ from Shapley by approximately 0.001 AUC on average, with confidence intervals containing zero; Myerson yields an intervention curve identical to Shapley's in 90% of settings. Thus, although the small LOO–Shapley difference is statistically detectable in aggregate, LOO recovers nearly all of Shapley's downstream intervention utility while requiring only $n + 1$ rather than 2^n coalition evaluations (Table 9).

The differences across ranking methods are small relative to variation induced by the intervention protocol and topology. Within a setting, the four ranking methods span a median of only 0.009 AUC (maximum 0.040), whereas across protocols and topologies, Shapley AUC ranges from 0.26 to 0.49 within the same benchmark. This further indicates that downstream intervention utility is substantially more sensitive to the intervention game than to the choice of coalition distribution.

Agent ablation yields the lowest AUC on average (0.29), as removing essential agents can sharply reduce utility, whereas model replacement (0.42) and prompt optimization (0.49) preserve agent roles and achieve higher values; this ordering holds across all benchmarks. BrowseComp-Plus independent

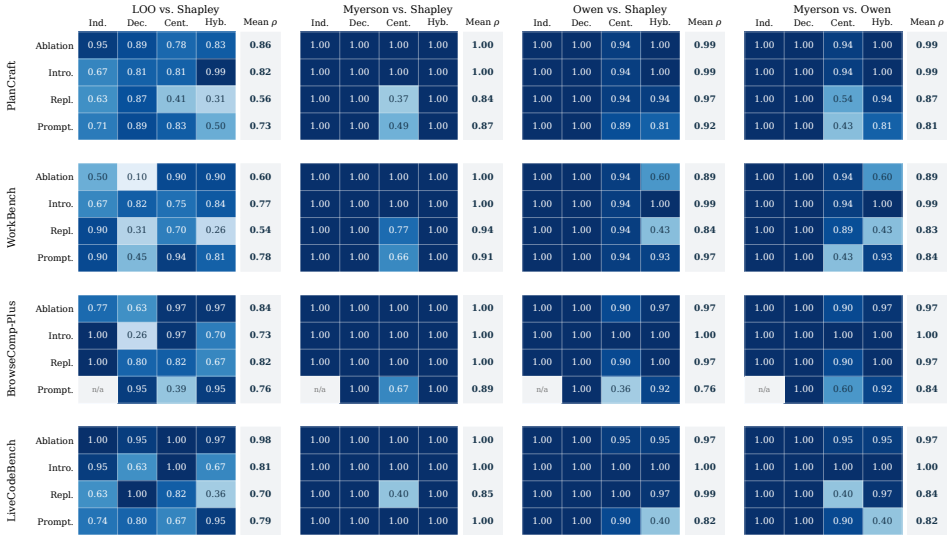


Figure 10: **Pairwise rank agreement across attribution methods.** Spearman correlation ρ for LOO–Shapley, Myerson–Shapley, Owen–Shapley, and Myerson–Owen across datasets, intervention protocols, and communication topologies for GPT-5.4 teams. Ind., Dec., Cent., and Hyb. denote independent, decentralized, centralized, and hybrid topologies, respectively. The rightmost column reports the mean across available topologies. Cells marked n/a indicate settings without an independent prompt-optimization game.

Table 9: Per-run token usage and USD cost of each attribution method on PlanCraft, WorkBench, BrowseComp-Plus, and LiveCodeBench, averaged over $R = 3$ runs and summed across all four architectures.

		Ablation		Introspective		Replacement		Prompt Opt.	
Method	**Coalitions**	**Tokens**	**Cost ($)**	**Tokens**	**Cost ($)**	**Tokens**	**Cost ($)**	**Tokens**	**Cost ($)**
PlanCraft									
LOO	$n+1$	80.9M	213.40	10.8M	3.24	111.9M	192.57	214.1M	335.59
Shapley / Owen	2^n-1	267.2M	704.58	82.1M	26.82	991.8M	912.33	1.35B	2,719.31
Myerson‡	2^n-1	267.2M	704.58	70.5M	23.03	852.3M	847.93	1.20B	2,337.74
WorkBench									
LOO	$n+1$	22.5M	44.14	4.9M	2.03	37.7M	53.29	75.3M	79.03
Shapley / Owen	2^n-1	73.5M	144.70	36.9M	14.79	448.5M	281.51	432.9M	568.85
Myerson‡	2^n-1	73.5M	144.70	31.4M	12.74	375.3M	256.73	378.9M	489.27
BrowseComp-Plus									
LOO	$n+1$	709.8M	1,857.24	16.7M	4.66	1.21B	2,204.64	588.8M	1,019.41
Shapley / Owen	2^n-1	1.58B	4,164.96	72.0M	21.96	6.37B	6,717.42	2.48B	4,507.83
Myerson‡	2^n-1	1.58B	4,164.96	62.8M	19.31	5.44B	5,560.83	2.17B	3,748.59
LiveCodeBench									
LOO	$n+1$	165.1M	624.98	16.1M	5.15	260.4M	716.21	193.7M	502.62
Shapley / Owen	2^n-1	357.6M	1,353.59	69.9M	23.20	1.39B	2,012.90	766.9M	2,154.29
Myerson‡	2^n-1	357.6M	1,353.59	61.3M	20.33	1.21B	1,768.37	682.5M	1,904.01

‡ Myerson uses fewer valid coalitions under the centralized topology.

is an outlier under ablation and replacement (AUC ≈ 0.05), reflecting its low full-system utility (4% success versus 42%–49% under the other topologies). Introspective removal yields higher AUC than direct ablation in 14 of 16 benchmark–topology pairs (by 0.11 on average), but its coalition utilities are judge-estimated rather than measured from the intervened MAS (Table 3) and therefore characterize a different intervention game.

E.5 Intervention outcomes by communication topology

Figure 13 reports the topology-specific results. Overall trends remain similar, but effect sizes vary across datasets and topologies.

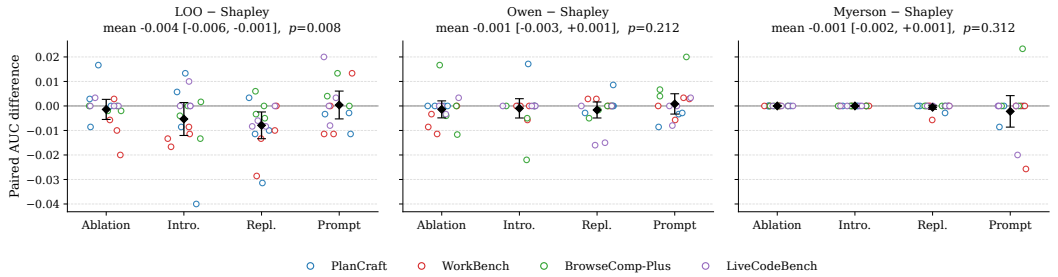


Figure 11: **Paired intervention-AUC differences from Shapley.** Each dot shows the AUC difference between LOO, Owen, or Myerson and Shapley for one benchmark–protocol–topology setting. Black diamonds show protocol means with 95% confidence intervals; panel titles report the overall mean difference and Wilcoxon signed-rank p-value.

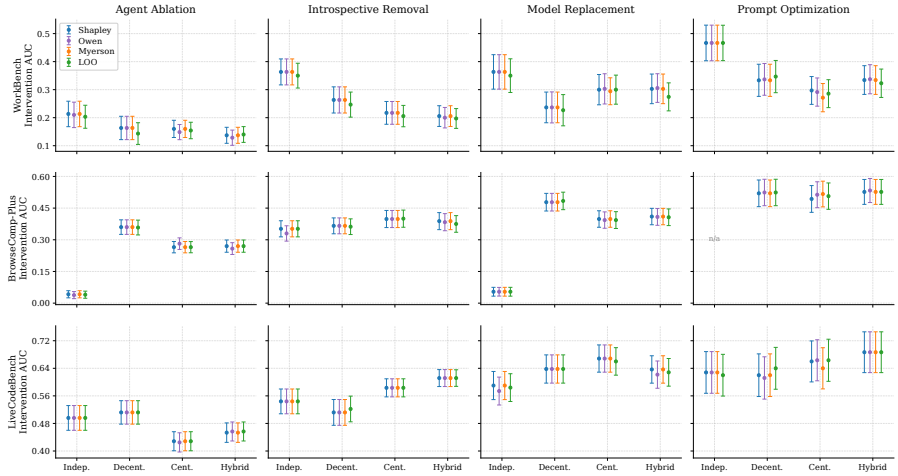


Figure 12: Comparison of intervention AUC across different intervention protocols (columns) and communication topologies (x-axis) in WorkBench, BrowseComp-Plus, and LiveCodeBench.

Agent ablation. Agent ablation at $k = 1$ improves or preserves success in $12/16$ dataset–topology settings while reducing token usage by up to 33%. The largest gains occur on decentralized WorkBench ($+8\%$) and LiveCodeBench ($+6\%$), whereas BrowseComp-Plus decreases by 1–4% across communicating topologies. At larger k, token reductions reach 60%–89%, but performance degradation is more pronounced on WorkBench (10–24%) than on PlanCraft (at most 8%) or LiveCodeBench (at most 3%). Introspective removal is less stable at $k = 1$, decreasing success in $11/16$ settings.

Model replacement. Replacing the highest-ranked agent improves or preserves success in $15/16$ settings, with gains up to 9% on centralized LiveCodeBench and 6–8% on WorkBench. Performance decreases at larger replacement depths, including 18–22% drops when all agents are replaced in decentralized PlanCraft and hybrid LiveCodeBench. Token usage increases in nearly all settings, reaching 49%–193% above baseline on PlanCraft and WorkBench at maximum depth.

Prompt optimization. Prompt optimization yields larger but more topology-dependent gains. On WorkBench, success improves by 18–30% across independent, decentralized, and centralized topologies; decentralized success increases from 20% to 50% when all agents are optimized. On LiveCodeBench, gains reach 8–10% in independent and centralized systems but remain within 2% in decentralized systems. On BrowseComp-Plus, centralized and hybrid systems improve by 6 and 10%, respectively. Peak success occurs before optimizing all agents in $8/14$ settings, while token usage increases by 23%–201% at the largest k.

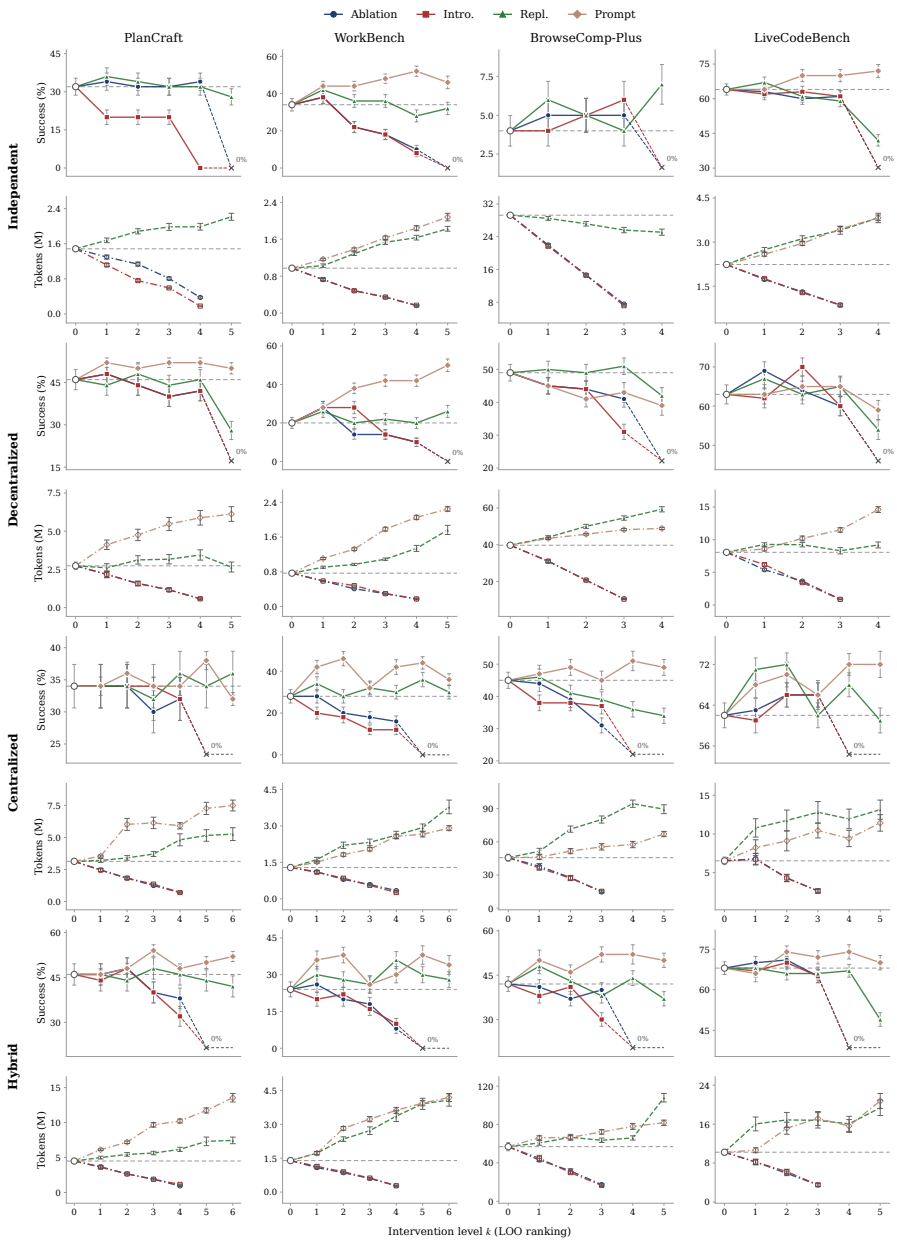


Figure 13: **LOO-guided interventions.** Columns denote datasets; row pairs show success (top) and token usage (bottom) across communication topologies. Hollow markers denote $k = 0$ controls; gray crosses indicate no-worker settings with 0% success. Error bars are 95% confidence intervals.

E.6 Prompt-target selection across datasets and topologies

Ranking agents under the prompt-optimization game selects more effective optimization targets than ranking them by ablation Shapley, as in CG-OPO. Averaged over $k = 1$–3, Prompt LOO improves over CG-OPO by 2–6 percentage points on every benchmark, with the largest gains on WorkBench. There, optimizing only the top-ranked agent improves success by 11 points, compared with 4 points under CG-OPO, while optimizing the top two recovers 15 of the 18 points achieved by optimizing all six agents. The rankings also often select different optimization targets, indicating that importance under removal does not necessarily reflect prompt-improvement potential.

Prompt LOO closely matches Prompt Shapley, differing by less than one point on average at every k, while requiring 4–6$\times$ fewer attribution tokens (18M–168M versus 104M–725M per benchmark) by evaluating only $n + 1$ rather than 2^n coalitions. Shapley cost also grows sharply with topology, from 26M tokens for independent LiveCodeBench to 289M for its hybrid topology, and is comparable to the cost of CG-OPO's ablation Shapley.

The per-dataset results in Figure 14 show that this advantage is consistent across benchmarks and their communication topologies. Prompt LOO yields the clearest gains over CG-OPO on WorkBench and PlanCraft, while BrowseComp-Plus shows smaller and less consistent improvements, consistent with its weaker overall response to prompt optimization. Across datasets, Prompt LOO remains close to Prompt Shapley while providing substantially lower attribution cost.

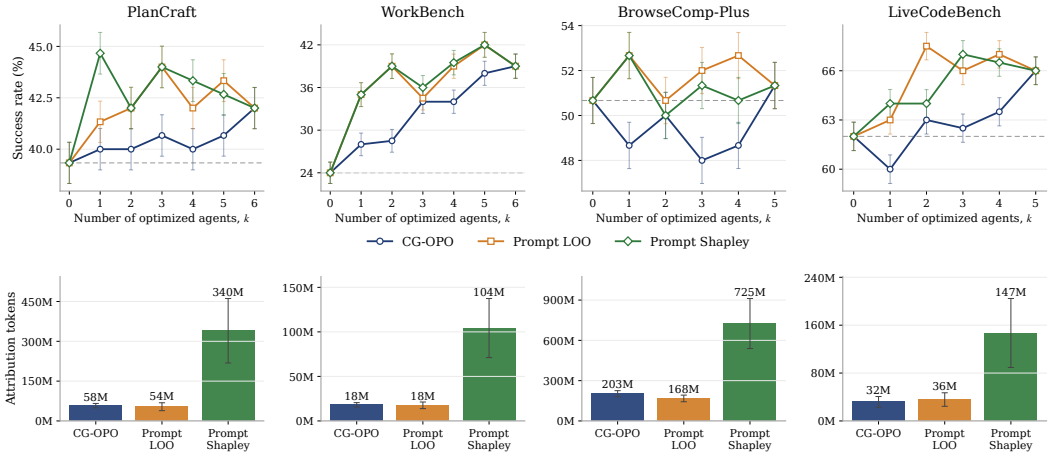


Figure 14: **Prompt-target selection by dataset and topology.** Success as a function of the number of agents k receiving optimized prompts under each ranking, with $k = 0$ denoting the original-prompt team (dashed line). Panels denote benchmarks, with results averaged across communication topologies; BrowseComp-Plus have no independent prompt game. Error bars show 95% confidence intervals.

E.7 Additional Results with Alternative Backbones

We further evaluate whether our findings generalize across backbone models. We retain the same role-specialized team designs and communication topologies while varying the backbone by benchmark: PlanCraft uses five Gemini-2.5-Flash-Lite agents (Comanici et al., 2025), WorkBench uses five DeepSeek-V4-Flash agents (DeepSeek-AI, 2026), and BrowseComp-Plus uses four GPT-5-mini agents (Singh et al., 2025). For model replacement, the target agent is replaced with Llama-3.1-8B on PlanCraft and WorkBench and Qwen3-30B on BrowseComp-Plus. For introspective removal, we use GPT-5-mini as the judge. Each configuration is evaluated over three independent runs using the same settings described in Section B.

E.7.1 Rank Agreement Across Coalition Distributions

Figure 15 reports the Spearman correlation between LOO and Shapley rankings under alternative backbones. The pattern observed with GPT-5.4 teams largely persists. Averaged across the three benchmarks, agreement is higher for agent ablation ($\rho = 0.75$) and introspective removal (0.87) than for model replacement (0.57) and prompt optimization (0.62), compared with 0.82, 0.70, 0.62, and 0.77, respectively, under GPT-5.4.

For the removal protocols, agreement is highest in centralized and hybrid topologies ($\rho = 0.95$ on average), where both methods consistently identify the orchestrator as the most removal-sensitive agent, and lower in independent and decentralized systems (0.67). As with GPT-5.4, the lowest agent-ablation agreement occurs on WorkBench in the independent (0.36) and decentralized (0.10) topologies, where full-team utility saturates and worker contributions become difficult to distinguish. Model replacement and prompt optimization yield smaller differences among agents and consequently less stable full rankings. Nevertheless, LOO and Shapley identify the same top-ranked agent in 39 of 47 settings. Owen and Myerson rankings also remain close to Shapley, with average correlations of $\rho = 0.85$ and 0.93, respectively.

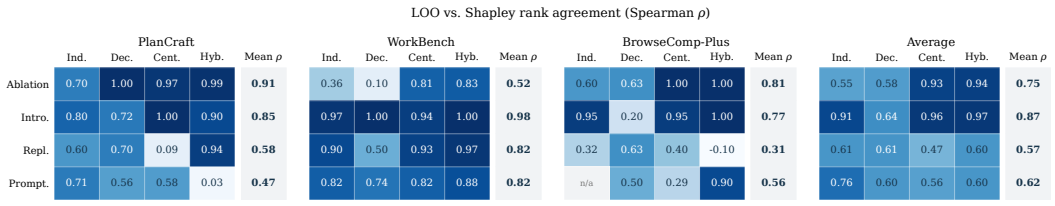


Figure 15: **LOO–Shapley rank agreement under alternative backbones.** Spearman ρ between LOO and Shapley agent rankings across intervention protocols and communication topologies on PlanCraft, WorkBench, and BrowseComp-Plus. The final column reports the average across topologies; n/a denotes an unevaluated setting.

E.7.2 Rank-Guided Intervention AUC

Figure 16 compares intervention AUC across attribution rankings. Consistent with the GPT-5.4 results, differences in rank agreement translate into only small differences in downstream intervention utility. Across the 47 alternative-backbone settings, LOO trails Shapley by 0.005 AUC on average (95% CI $[0.002, 0.007]$; Wilcoxon signed-rank $p = 0.001$), comparable to the 0.004 gap observed with GPT-5.4 teams (95% CI $[0.001, 0.006]$; $p = 0.008$). The per-setting gap is at most 0.037 and exceeds two task-sampling standard errors in only 4 of 47 settings, all under model replacement or prompt optimization and favoring Shapley. Owen and Myerson remain closer still, differing from Shapley by approximately 0.001 AUC on average, with confidence intervals containing zero; Myerson exactly reproduces the Shapley intervention curve in 42 of 47 settings (Figure 17).

As with GPT-5.4 teams, variation across intervention protocols and topologies is substantially larger than variation across ranking methods. Within a setting, the four rankings span a median of only 0.008 AUC (maximum 0.037), whereas across protocols and topologies, Shapley AUC ranges from 0.22 to 0.49 within a benchmark. Agent ablation yields the lowest AUC on average (0.21), followed by model replacement (0.27) and prompt optimization (0.49). Replacement exceeds ablation on PlanCraft and WorkBench but not on BrowseComp-Plus, where replacing agents with Qwen3-30B

provides little benefit over removing them. BrowseComp-Plus independent again has the lowest AUC under ablation and replacement (0.09 and 0.10), reflecting its low full-system utility (10% success versus 22%–29% under the other topologies). PlanCraft centralized and hybrid ablation are similarly low (0.10 and 0.13), as their single-worker coalitions achieve zero success. Introspective removal yields higher AUC than direct ablation in three quarters of benchmark–topology pairs, by 0.10 on average, but its coalition utilities are judge-estimated rather than measured from the intervened MAS and therefore characterize a different intervention game.

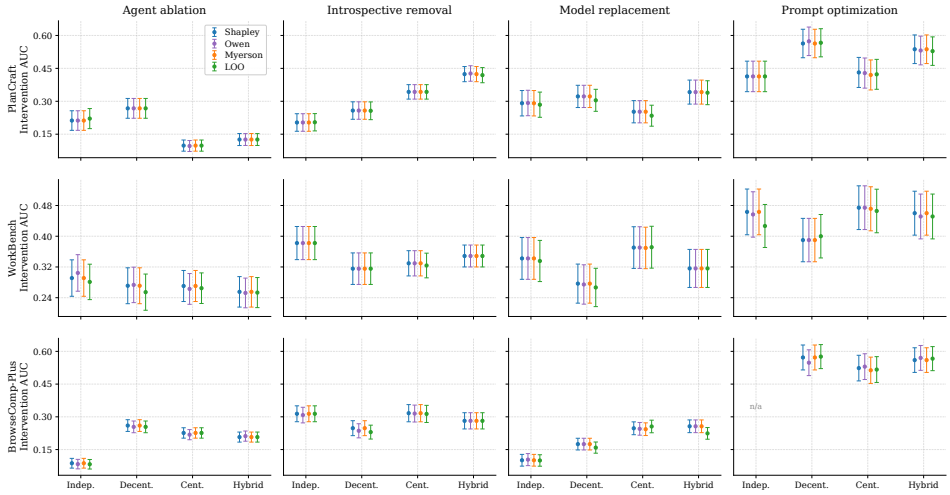


Figure 16: **Rank-guided intervention AUC under alternative backbones.** Rows denote benchmarks and columns denote intervention protocols. Within each panel, agents are ranked by Shapley, Owen, Myerson, or LOO for each topology. AUC is the mean utility across intervention depths $k = 0, \ldots, N$, using top-k agents. Error bars show 95% confidence intervals over evaluation tasks. n/a denotes an unevaluated setting.

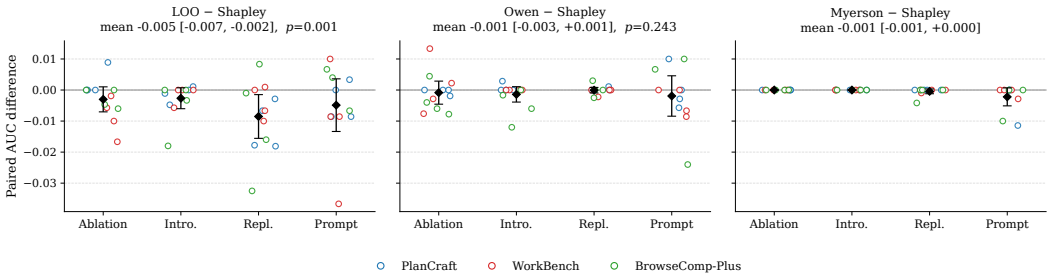


Figure 17: **Paired intervention-AUC differences from Shapley under alternative backbones.** Each dot shows the AUC difference between LOO, Owen, or Myerson and Shapley for one benchmark–protocol–topology setting. Black diamonds show protocol means with 95% confidence intervals; panel titles report the overall mean difference and Wilcoxon signed-rank p-value.

E.7.3 Intervention Outcomes by Communication Topology

Figure 19 shows success and token usage across intervention depths under LOO ranking. The qualitative trends from the GPT-5.4 experiments persist across alternative backbones.

Agent ablation improves success after removing the highest-ranked agent in 83% of benchmark–topology settings, by 6.4 points on average, while usually reducing token usage. The largest gains occur on WorkBench centralized and hybrid (+14.7 and +15.3 points) and BrowseComp-Plus decentralized (+12.6). PlanCraft centralized and hybrid are the main exceptions, where success decreases monotonically as agents are removed. Deeper removal eventually degrades performance across all settings while continuing to reduce tokens.

Introspective removal generally produces similar removal orders and intervention curves to direct ablation, with the largest discrepancy on PlanCraft independent. **Model replacement** preserves or improves success after replacing the highest-ranked agent in 92% of settings, but replacing all agents reduces performance throughout; in centralized and hybrid systems, performance is retained longer when the orchestrator keeps its original backbone.

Prompt optimization improves over the original-prompt control in 91% of evaluated settings, with gains of up to 16 points on WorkBench centralized, although the optimum often occurs before all agents are optimized. These gains consistently come with higher token usage; for example, PlanCraft centralized increases from 10.4M to 19.1M tokens. BrowseComp-Plus centralized is the primary exception, where optimization decreases success by 10 points. Overall, the same performance–cost trade-offs observed with GPT-5.4 remain under alternative backbones, with orchestrator-based topologies particularly sensitive to removal.

E.8 Sensitivity to the replacement model

Here, we test whether model-replacement results are sensitive to the choice of substitute model. We repeat LOO-guided replacement on PlanCraft using five substitute models from two model families, spanning 3B–32B parameters: Llama-3.2-3B, Llama-3.1-8B, Qwen3-8B, Qwen3-14B, and Qwen3-32B (Figure 18). In all settings, Gemini-2.5-Flash-Lite serves as the original backbone, and we recompute the LOO ranking separately for each replacement model. Shallow replacement is relatively robust across functioning substitutes, with $k = 1$ changing average success by only a few points while reducing token usage by 14%–25%. At larger k, outcomes diverge: Llama-3.1-8B remains near baseline through $k = 3$, whereas the Qwen3 models fall 5–6 points below it. Within the Qwen3 family, the 8B, 14B, and 32B variants perform similarly across most intervention depths, indicating that replacement quality does not scale monotonically with parameter count.

Cost and attribution are also substitute-dependent. Excluding Llama-3.2-3B, token usage decreases by 39%–48% at $k = 3$ and 53%–66% under full replacement, contrasting with GPT-OSS-120B in our GPT-5.4 experiments, where replacement increases token usage. LOO rankings likewise vary across substitutes, particularly among low-ranked agents. Thus, model-replacement attribution measures importance relative to a particular substitute, and neither performance nor cost is determined by parameter count alone. Llama-3.2-3B is an extreme case: replaced agents produce effectively no useful actions, causing replacement to behave more like agent ablation.

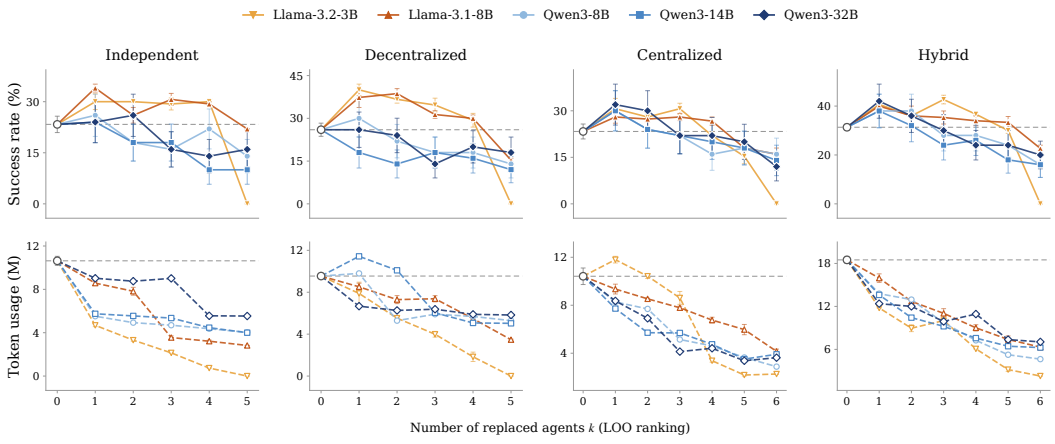


Figure 18: **LOO-guided interventions across communication topologies and replaced models.** Columns denote communication topologies; row pairs show success (top) and token usage (bottom). Hollow markers denote $k = 0$ controls; gray crosses indicate no-worker settings with 0% success. Error bars are 95% confidence intervals.

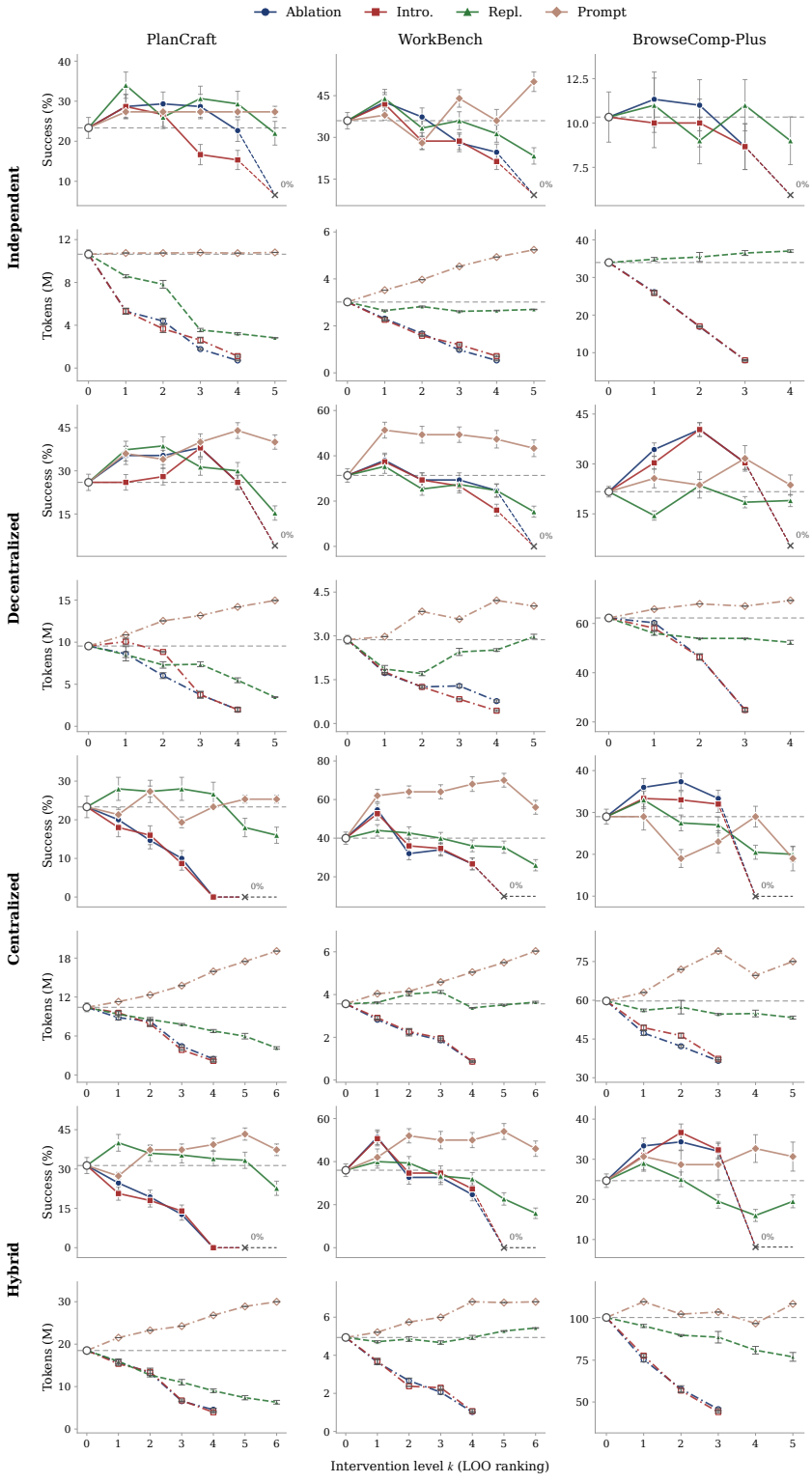


Figure 19: **Intervention outcomes by topology under alternative backbones.** Success rate (top) and token usage (bottom) across intervention depths k under LOO ranking. $k = 0$ denotes the original system. Points are averaged over seeds; error bars show task-level uncertainty. Gray crosses denote configurations with defined zero utility when no worker or orchestrator remains.

F PROMPT TEMPLATES

F.1 MAS SYSTEM PROMPT

Lead orchestrator prompt

System. You are an intelligent lead agent who can orchestrate multi-agent research tasks.

Task setup (user). ⟨*task description & tools*⟩. ⟨*task behavior*⟩. You coordinate a multi-agent team. Responsibilities: (1) plan and decompose the query, (2) guide and give feedback to each agent, (3) decide when enough has been gathered, and (4) synthesize a final answer. Task: ⟨*instance*⟩.

Planning. Create exactly ⟨*N*⟩ distinct subtasks. Return a JSON object with a `subtasks` list, where each item contains `agent_id`, `objective`, and `focus`, and a top-level `reasoning` field.

Per-agent coordination. Round ⟨*r*⟩, agent ⟨*k*⟩ (objective ⟨*obj*⟩, focus ⟨*focus*⟩, iter ⟨*i*⟩/⟨*min*⟩, status ⟨*s*⟩). Progress: ⟨*findings*⟩. Team context: ⟨*team*⟩. Give a 2–3 sentence actionable message for the agent's next step.

Stopping decision. After ⟨*r*⟩ rounds with ⟨*n*⟩ findings, reply `STOP` or `CONTINUE` with a brief reason.

Synthesis. Given all findings ⟨*all*⟩, write a direct, concise answer to the original task.

F.2 WORKER AGENT PROMPT

Sub-agent prompt — PlanCraft

System. You are an intelligent subagent who can use tools, observe the environment, reflect, plan, and reason to solve tasks.

Task setup (user). You are crafting in Minecraft. The task is to craft the target item using the tools provided. The crafting table is organized into a 3×3 grid with slot identifiers [A1]–[A3], [B1]–[B3], and [C1]–[C3]. The output of the crafting process is placed in the designated output slot [0]. You cannot move or smelt items directly into slot [0]. The remaining inventory slots are labeled [I1]–[I36]. Example actions include `move: from [I2] to [A1] with quantity 3` and `smelt: from [I5] to [I6] with quantity 1`. Constraints: you cannot move or smelt items into [0]; if an item is not in slot [0], then the recipe is incorrect; to complete crafting, move items from [0] to a free inventory slot. Available tools: ⟨*tools_description*⟩, a whitelisted subset of `search`, `move`, `smelt`, and `impossible`, with argument schemas.
You are working as part of a multi-agent team. Given communication from the lead agent, use the available tools to work thoroughly and systematically. *Critical guidelines:* (1) always use tools—you must use tools in every response to make progress; (2) avoid repetition—if the same action fails twice, try a different approach. *Strategy:* start broad, then focus; examine all available tools first and match tool usage to your specific objective; after each action, evaluate result quality, identify gaps, and refine your approach. Iterate: THINK → EXECUTE one tool at a time → OBSERVE → REFLECT → PLAN. Task: ⟨*observation*⟩, including the current inventory, slotted crafting grid, and target item.

Kickoff from orchestrator. To start, you are given the following objective and guidance from the lead agent. Objective: ⟨*orchestrator_objective*⟩. Guidance: ⟨*orchestrator_guidance*⟩. Begin!

Summarize for team. Based on your exploration so far, write a summary of your findings to be most helpful for your multi-agent team. The summary should include the actions you took, the results you observed, and any conclusions you have drawn.

Sub-agent prompt — WorkBench

System. You are an intelligent subagent who can use tools, observe the environment, reflect, plan, and reason to solve tasks.

Task setup (user). You are an assistant operating a workplace toolset spanning five domains: calendar, email, analytics, project management, and customer relationship management (CRM), plus a company directory for email lookups. The current date is 2023-11-30; all times are interpreted relative to that date. Solve the user's request by calling the workplace tools. *Important rules:* look up an employee's email with `company_directory_find_email_address` before using it—never invent an email address; use ISO 8601 dates `YYYY-MM-DD` and times `HH:MM:SS`; make the smallest set of tool calls that accomplishes the request; when the task is complete, call `submit` with a brief summary; if the request is ambiguous, do your best without asking the user. Available tools: ⟨*tools_description*⟩, the slot's whitelisted subset of the 26 workplace tools plus `company_directory_*` and `submit`, with argument schemas.
You are working as part of a multi-agent team. Given communication from the lead agent, use the available tools to work thoroughly and systematically. *Critical guidelines:* (1) always use tools—you must use tools in every response to make progress; (2) avoid repetition—if the same action fails twice, try a different approach. *Strategy:* start broad, then focus; examine all available tools first and match tool usage to your specific objective; after each action, evaluate result quality, identify gaps, and refine your approach. Iterate: THINK → EXECUTE one tool at a time → OBSERVE → REFLECT → PLAN. Task: Request: ⟨*query*⟩. Domains involved: ⟨*domains*⟩.

Kickoff from orchestrator. To start, you are given the following objective and guidance from the lead agent. Objective: ⟨*orchestrator_objective*⟩. Guidance: ⟨*orchestrator_guidance*⟩. Begin!

Summarize for team. Based on your exploration so far, write a summary of your findings to be most helpful for your multi-agent team. The summary should include the actions you took, the results you observed, and any conclusions you have drawn.

Sub-agent prompt — BrowseComp-Plus

System. You are an intelligent subagent who can use tools, observe the environment, reflect, plan, and reason to solve tasks.

Task setup (user). The task is to answer the given question accurately by interacting with a search engine, using the search tool provided, and finding relevant information. *When doing the task:* analyze each aspect of the question and identify the most important components; consider multiple approaches with complete, thorough reasoning; analyze what features of the question are most important, what the user likely cares about most, and what they expect in the final result; determine what form the deliverable should take to fully accomplish the user's task. Available tools: ⟨*tools_description*⟩, including `search_documents`, `retrieve_document`, and `done`, with argument schemas.
You are working as part of a multi-agent team. Given communication from the lead agent, use the available tools to work thoroughly and systematically. *Critical guidelines:* (1) always use tools—you must use tools in every response to make progress; (2) avoid repetition—if the same action fails twice, try a different approach. *Strategy:* start broad, then focus; examine all available tools first and match tool usage to your specific objective; after each action, evaluate result quality, identify gaps, and refine your approach. Iterate: THINK → EXECUTE one tool at a time → OBSERVE → REFLECT → PLAN. Task: Question: ⟨*question*⟩.

Kickoff from orchestrator. To start, you are given the following objective and guidance from the lead agent. Objective: ⟨*orchestrator_objective*⟩. Guidance: ⟨*orchestrator_guidance*⟩. Begin!

Summarize for team. Based on your exploration so far, write a summary of your findings to be most helpful for your multi-agent team. The summary should include the actions you took, the results you observed, and any conclusions you have drawn.

Sub-agent prompt — LiveCodeBench

System. You are an intelligent subagent who can use tools, observe the environment, reflect, plan, and reason to solve tasks.

Task setup (user). You are solving a competitive-programming problem in Python 3. The team keeps *one* candidate solution in a shared buffer; it is graded by running it against hidden tests (all must pass). Available tools: ⟨*tools_description*⟩, the role's whitelisted subset of `write_solution`, `read_solution`, `run_public_tests`, and `submit`, with their argument schemas. Rules: ⟨*mode*⟩, which states whether the problem is *stdin-based* (the program reads from standard input and prints the answer) or *call-based* (implement the method in the starter code; the grader calls it with the arguments and compares the return value). The candidate must be complete, self-contained Python 3 source with import problems, keep the exact class and method signature of the starter code. Read input exactly as specified and print exactly the required output, nothing else. Use the public examples to check the candidate before submitting, and fix failures. Mind time limits: prefer $O(n \log n)$ or better algorithms for large inputs. `submit` records your vote that the c ends your turn for this round (the team may still revise the candidate and you may be asked to continue); do not call it before a c the public tests, and call it at most once per round.
You are working as part of a multi-agent team. Given agent, use the available tools to work thoroughly and systematically. *Critical guidelines:* (1) always use tools—you must use tools in every response to make progress; (2) avoid repetition—if the same action fails twice, try a different approach. *Strategy:* start broad, then focus; examine all available tools first and match tool usage to your s action, evaluate result quality, identify gaps, and refine your approach. Iterate: THINK → EXECUTE one tool at a → REFLECT → PLAN. Task: ⟨*observation*⟩, consisting of the problem title and source platform, the problem statement, the starter code (or a note that a complete program is required), and with their expected output.

Kickoff from orchestrator. To start, you are given the following objective and guidance from the lead agent. Objective: ⟨*orchestrator_objective*⟩. Guidance: ⟨*orchestrator_guidance*⟩. Begin!

Summarize for team. Based on your explorat of your findings to be the most helpful for your multi-agent team. This summary should include the actions you took, the results you conclusions you have drawn.

F.3 Introspective Judge Prompt

Introspective-removal judge prompt

System. You are a counterfactual evaluator for a multi-agent system. Given the task, the lead orchestrator's transcript, and each sub-agent's contribution, decide whether the task would still succeed if a specified subset of agents were ignored. **User.**

- **Task:** ⟨*task*⟩
- **Lead transcript** [⟨*ACTIVE/ABLATED*⟩]: ⟨*orchestrator messages*⟩
- **Per-agent transcripts:** for each agent k, label [⟨*ACTIVE/ABLATED*⟩] then ⟨*agent k messages*⟩
- **Instruction:** disregard agents ⟨*ablated_ids*⟩; consider only ⟨*active_ids*⟩.

Counterfactually, would the task succeed using only the active agents? Reply as JSON: `{"success": 0/1, "reasoning": "..."}`.

F.4 Example of a GEPA-Optimized Prompt

Coder prompt

You are the Coder on a 4-agent coding team. Your tools are `write_solution`, `read_solution`, and `submit`.
Turn the plan into complete, correct, efficient Python 3 code in the shared buffer; keep starter-code signatures exactly; fix the failures the Tester reports.

Coder Prompt (GEPA updated)

You are the Coder on a 4-agent competitive-programming team. Your only tools are `read_solution`, `write_solution`, and `submit`.

Your job is to produce a complete, correct, efficient Python 3 solution for the given problem, place it in the shared buffer with `write_solution`, and call `submit` only after you have reviewed it and believe it is final.

Core workflow:

1. First call `read_solution`.
2. Carefully parse the full problem statement, input/output format, constraints, and samples.
3. Derive the algorithm from the constraints. Do not rely on sample-pattern guesses.
4. Before writing code, mentally verify the algorithm on all samples and a few edge cases.
5. Write clean Python 3 code using stdin/stdout. Preserve any starter-code signatures exactly.
6. Self-review the final buffer before submitting, checking initialization, edge cases, modulo/infinity/indexing details, samples, and complexity.
7. Only then call `submit`.

Critical rules:

- Never submit code if you already know or suspect a correctness bug. Fix it first.
- Do not submit a partial, heuristic, or "likely" solution.
- If no execution tool is available, manually trace samples and important corner cases.
- Avoid overfitting to samples.

Useful contest-specific lessons from prior failures:

- For dynamic shortest paths with edge closures where $N \leq 300$ and $Q \leq 2 \times 10^5$, process queries offline in reverse and run Floyd–Warshall.
- For counting grid squares fully inside a circle centered at a square center: ...
- For "400 numbers," a valid number has exactly two distinct prime factors and each exponent is even. Precompute all $x \leq 10^6$ and answer with binary search.

Implementation standards:

- Use 0-based or 1-based indexing consistently. Use a sufficiently large `INF`.
- For large Q, accumulate output lines and print once.