DISENTANGLING AGENT CONTRIBUTION IN LLM MULTI-AGENT SYSTEMS THROUGH CONTROLLED INTERVENTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Identifying which agent actually drives the success of a large language model (LLM)-based multi-agent system remains an open challenge. Aggregate performance cannot answer this question, and our results show that attribution does not simply identify the most capable reasoner. We introduce MASCONTRIBUTIONBENCH, covering seven architectures and seven task suites, and combine leave-one-out (LOO), sampled Shapley, and sampled Banzhaf attribution with controlled interventions on topology, role prompts, and runtime permissions. Complete MAS outperform single-agent and solo-role baselines, while the aggregate difference from random teams remains unresolved; collaboration gains alone therefore do not validate a prescribed organization. More strikingly, changing validated static predecessor sets reverses an unchanged position's mean LOO sign with modest team-utility change. Exchanging functional prompts moves credit to unedited roles without improving the team, demonstrating that contribution does not simply follow role function. Changing Coder write access leaves 87.8% of outcomes unchanged, while granting final-answer authority to Verifier produces no Verifier-selected outputs in all runs. These findings reveal that measured contribution conflates useful reasoning, routing leverage, and control of the evaluator-visible output. Contribution therefore measures the counterfactual fragility of an execution protocol, rather than an intrinsic property of an agent or role.

1 INTRODUCTION

Large language model (LLM)-based multi-agent systems (MAS) organize multiple model calls into workflows with specialized roles, communication structures, and collaboration protocols (Li et al., 2023; Wu et al., 2023; Qian et al., 2024). Recent systems further automate workflow construction and orchestration (Qian et al., 2025; Zhang et al., 2025a; Dang et al., 2026; Yang et al., 2026a). Despite this growing structural complexity, MAS are still evaluated primarily by end-task performance. Such evaluation tells us whether a system succeeds, but not why: it cannot identify which positions create utility, whether a prescribed organization improves on a comparable team, or whether an observed gain comes from reasoning, information routing, or control of the final output.

Understanding these mechanisms is important for more than interpreting a score. System designers need to know which agents can be removed, where redundancy is useful, whether an expensive role improves the answer, and whether a workflow will remain effective after its topology or permissions change. A misleading attribution can produce the wrong design decision. For example, a Finalizer may appear indispensable because it performs valuable reasoning, but it may instead be indispensable only because it controls the sole evaluator-visible output. Conversely, a role with negative attribution may be redundant in one workflow rather than intrinsically harmful.

We study *agent contribution* as counterfactual marginal utility: how the evaluated system outcome changes when an agent position is present rather than absent under a specified execution context. This quantity is difficult to interpret for three reasons. First, removing an agent changes not only its reasoning but also the information available downstream. Second, attribution depends on the counterfactual context: leave-one-out (LOO) measures dependence in the complete workflow, whereas Shapley and Banzhaf values average marginal utility over coalitions. Third, role prompts, graph po-

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.

sitions, runtime permissions, and output selection are entangled. A high score can therefore reflect useful reasoning, routing leverage, or authority over the interface observed by the evaluator.

We introduce MASCONTRIBUTIONBENCH, a controlled benchmark designed to separate these mechanisms. It evaluates seven MAS architectures and seven task suites with matched single-agent, solo-role, and random-team baselines. The benchmark combines LOO, sampled Shapley, and sampled Banzhaf attribution with interventions that modify communication topology, exchange functional role prompts, or change runtime permissions while holding the model, tasks, seeds, and evaluation protocol fixed (Figure 1).

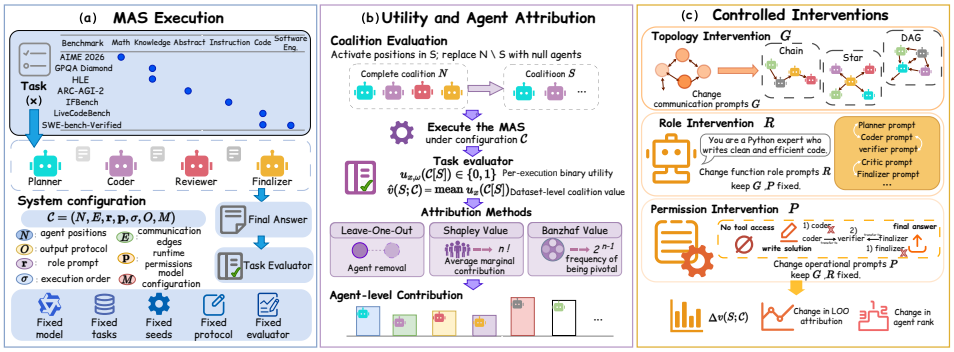


Figure 1: Overview of MASCONTRIBUTIONBENCH. A multi-agent system executes task x under configuration $\mathcal{C} = (N, E, \mathbf{r}, \mathbf{p}, \sigma, O, M)$. Coalition-based methods estimate the marginal utility of agent positions, while controlled interventions modify communication topology, role prompts, or runtime permissions to identify the mechanisms that shape contribution.

Three findings emerge. First, *collaboration gains do not validate a prescribed organization*: complete MAS outperform non-collaborative baselines, while their aggregate difference from random teams remains unresolved. Second, *contribution is not portable across execution contexts*: validated Chain–DAG rerouting reverses an unchanged position's aggregate mean LOO sign, while prompt swaps move credit to unedited positions with little mean utility change. Third, *nominal permission is not realized control*: restricting an artifact channel need not prevent useful work from propagating through messages, while final-answer authority matters only when it is both irreplaceable and selected by the output protocol.

Our central result is therefore interpretive: *contribution scores do not locate intelligence; they locate counterfactual fragility in a particular execution protocol*. High attribution may reflect useful reasoning, but it may also reflect leverage over an information route or monopoly over the evaluator-visible output. Likewise, negative attribution may reflect redundancy or coordination overhead rather than intrinsic harmfulness.Our contributions are:

- Introduces MASCONTRIBUTIONBENCH, a controlled evaluation framework spanning seven MAS architectures and seven task suites, with matched system baselines and audited execution traces.
- Distinguishes LOO complete-workflow dependence from coalition-averaged Shapley and Banzhaf utility.
- Disentangles reasoning, routing leverage, and output authority through controlled interventions, reframing contribution as execution-context dependence rather than intrinsic role capability.

2 METHOD

2.1 PROBLEM FORMULATION

Let $N = \{1, \ldots, n\}$ denote agent *positions*, $x \in \mathcal{D}$ a task, and $\omega \in \Omega$ an execution seed. We represent a system as

$$\mathcal{C} = (N, E, \mathbf{r}, \mathbf{p}, \sigma, O, M), \qquad h_i = F_M(x, r_i, p_i, \{h_j : (j, i) \in E,\ \sigma(j) < \sigma(i)\}; \omega_i)\,, \tag{1}$$

where E is the directed communication edge set, r_i the functional prompt at position i, p_i its runtime permissions, σ the execution order, O the output protocol, and M the shared model configuration. Thus h_i is generated from the task and the outputs visible from previously executed in-neighbors. Separating E from σ is important because our topology intervention rewires information flow while retaining a common single-pass execution order.

For an active coalition $S \subseteq N$, $\mathcal{C}[S]$ replaces every $i \notin S$ by a fixed, task-independent null response while preserving its position, edges, order, and permissions. Let $Y_{x\omega}(\mathcal{C}[S])$ be the evaluator-visible output produced by O. With task evaluator $e_x(\cdot) \in [0,1]$, define

$$\widehat{v}(S;\mathcal{C}) = \frac{1}{|\mathcal{D}||\Omega|} \sum_{x\in\mathcal{D}} \sum_{\omega\in\Omega} \underbrace{\mathbf{1}\{Y_{x\omega}(\mathcal{C}[S]) \neq \perp\} e_x(Y_{x\omega}(\mathcal{C}[S]))}_{u_{x\omega}(\mathcal{C}[S])}, \tag{2}$$

where $\perp$ denotes the absence of an output accepted under the permission and submission protocol. This validity gate distinguishes solution quality from authority to expose a solution to the evaluator. For the A2/A3 `final_answer` permission contrast, we exclude ARC-AGI-2 from the primary utility analysis because its evaluator path can score a retained artifact even when O selects no authorized final answer. We retain those runs only as an evaluator-path diagnostic in Appendix D.3.

2.2 Attribution Estimands

For position i and context $S \subseteq N \setminus \{i\}$, write its marginal utility as $m_i(S) = \widehat{v}(S \cup \{i\}; \mathcal{C}) - \widehat{v}(S; \mathcal{C})$. The three attribution quantities are compactly defined by

$$\widehat{\Delta}_i^{\mathrm{LOO}} = m_i(N \setminus \{i\}), \qquad \widehat{\phi}_i = \frac{1}{K}\sum_{k=1}^{K} m_i(P_i(\pi_k)), \qquad \widehat{\beta}_i = \frac{1}{K}\sum_{k=1}^{K} m_i(S_{ik}), \tag{3}$$

where $P_i(\pi_k)$ contains the positions preceding i in a uniformly sampled permutation π_k, and each S_{ik} includes every $j \neq i$ independently with probability $1/2$. We use $K = 8$ for both sampled Shapley (Shapley et al., 1953) and sampled Banzhaf (Banzhaf III, 1964). LOO therefore measures full-system necessity, whereas the latter two average marginal utility over different coalition distributions. Team utility is $\widehat{v}(N;\mathcal{C})$.

2.3 Controlled Interventions

We intervene on one component of the reference configuration $\mathcal{C}_0 = (N, E_0, \mathbf{r}_0, \mathbf{p}_0, \sigma, O, M)$ at a time:

$$\mathcal{C}^I = \begin{cases} (N, E_g, \mathbf{r}_0, \mathbf{p}_0, \sigma, O, M), & I = \mathrm{topology}(g), \\ (N, E_0, \tau_{ab}(\mathbf{r}_0), \mathbf{p}_0, \sigma, O, M), & I = \mathrm{role}(a,b), \\ (N, E_0, \mathbf{r}_0, \mathbf{p}_0^{k:i\to j}, \sigma, O, M), & I = \mathrm{permission}(k; i \to j). \end{cases} \tag{4}$$

Here, $g \in \{\text{Chain, Star, DAG}\}$ and τ_{ab} exchanges only the role prompts at positions a and b. For an operational permission $k \in \{\texttt{write_solution}, \texttt{final_answer}\}$, i denotes the designated source position whose permission is modified, and $j \in N \cup \{\varnothing\}$ denotes the recipient position. We define the modified permission map as

$$\mathbf{p}_0^{k:i\to j}(\ell, k') = \begin{cases} 0, & (\ell,k') = (i,k), \\ 1, & j \neq \varnothing \text{ and } (\ell,k') = (j,k), \\ \mathbf{p}_0(\ell,k'), & \text{otherwise.} \end{cases} \tag{5}$$

Thus, $j = \varnothing$ denotes revocation at source position i, whereas $j \in N$ denotes a source-to-recipient transfer: permission k is revoked from i and granted to j, while the permissions of all other positions remain unchanged from $\mathbf{p}_0$. For transfers, $j \neq i$ and $\mathbf{p}_0(i,k) = 1$. We evaluate the baseline $\mathbf{p}_0$, revocation of `write_solution` from Coder and `final_answer` from Finalizer, and source-to-recipient transfers of these permissions from Coder and Finalizer, respectively, to Verifier. Permissions are enforced by the runtime at artifact-writing and answer-submission time rather than through prompt instructions alone.

For each intervention, we compare the resulting system with the reference configuration $\mathcal{C}_0$ on identical task instances and random seeds. We quantify its team-level and position-level effects as

$$\widehat{\delta}^I_{\text{team}} = \widehat{v}(N;\mathcal{C}^I) - \widehat{v}(N;\mathcal{C}_0), \qquad \widehat{\delta}^I_{i,\text{LOO}} = \widehat{\Delta}^{\text{LOO}}_i(\mathcal{C}^I) - \widehat{\Delta}^{\text{LOO}}_i(\mathcal{C}_0). \tag{6}$$

Topology intervention. Chain and DAG vary only in validated static predecessor sets while preserving roles, execution order, permissions, and output selection. A configured-Star condition is retained as an executor diagnostic because its empty-upstream fallback exposes accumulated global history (Appendix D.1).

Functional role-prompt intervention. Swapping functional prompts between designated positions tests prompt sensitivity at fixed graph positions. We hold permissions, model, and output protocol constant; this intervention changes function, not authority.

Permission intervention. Two runtime-enforced permissions control access to the intermediate solution artifact (`write_solution`) and evaluator-visible final submission (`final_answer`). We evaluate revocation from the designated source position and source-to-recipient transfer to Verifier. Under a transfer, the permission is revoked from Coder or Finalizer and granted to Verifier, while permissions of all other positions remain unchanged. The runtime blocks unauthorized actions; when no authorized final output exists, equation 2 equals zero.

3 Experiments

3.1 Experimental Setup

Task suites.

- **Benchmarks.** We evaluate MASContributionBench on seven benchmarks: AIME 2026 (Zhang & Math-AI, 2026), GPQA Diamond (Rein et al., 2023), HLE (Phan et al., 2025), ARC-AGI-2 (Chollet et al., 2025), IFBench (Pyatkin et al., 2026), LiveCodeBench (Jain et al., 2025), and SWE-bench Verified (Jimenez et al., 2024; Chowdhury et al., 2024).
- **Evaluation protocol.** The first five benchmarks use fixed seeded subsets of 30 tasks with local evaluators. We use all 701 LiveCodeBench tasks for full-system evaluation and a matched 30-task subset for baseline comparison, while SWE-bench Verified is evaluated on 30 tasks using its official harness.

MAS architectures. Seven predefined MAS architectures vary in role organization and communication topology. Figure 2 summarizes their compact descriptions; complete role specifications, graph definitions, and architecture diagrams are deferred to Appendix A.2.

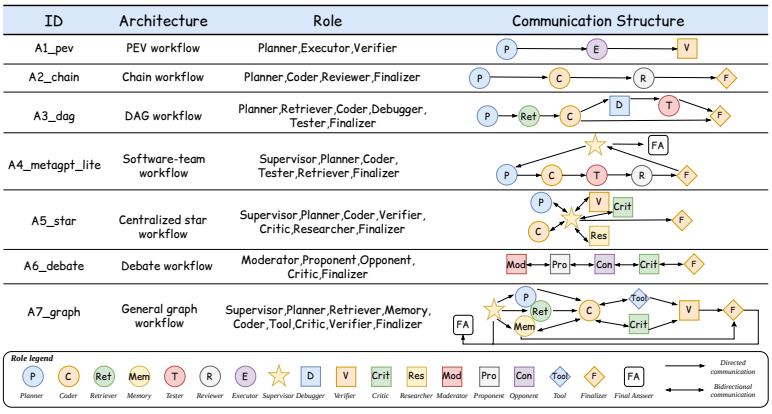


Figure 2: Overview of the seven MAS architectures.

Baselines. We compare the full MAS architectures with the following performance baselines:

- **Single-agent baseline.** A single general-purpose agent directly solves each task.

- **Solo-role baselines.** A single agent is prompted with one of the available functional roles, including Planner, Coder, Debugger, Verifier, Critic, Researcher, Supervisor, and Finalizer; see Appendix A.3 for the complete role definitions and prompt specifications.
- **Random-team baseline.** For each task, sample identifier, and execution seed, we construct a five-role team comprising a mandatory Finalizer and four role IDs sampled without replacement from the remaining pool. Roles execute once in the sampled order with access to all preceding messages, and the Finalizer output is evaluated. Three sample identifiers and three seeds produce nine runs per task, controlling team size while randomizing role composition and execution order.

Common execution protocol. All experiments use the same underlying model, decoding configuration, random-seed protocol, execution environment, and task evaluators. Unless otherwise specified, we use three random seeds with temperature 0.2 and a maximum generation length of $2{,}048$ tokens. Code and structured outputs are executed in a Docker sandbox, and each run records the outputs, traces, tool calls, token usage, runtime, cost, and failure type. Direct comparisons reuse the same task instances and random seeds.

3.2 Does MAS Create a Meaningful Attribution Setting?

We first establish that the benchmark yields a meaningful collaborative setting before assigning utility to individual agents or mechanisms. We compare seven complete MAS architectures with single-agent, solo-role, and random-team baselines under matched model, task, seed, decoding, and evaluation settings. The primary analysis uses the five locally scorable benchmarks in Section 3.1. LiveCodeBench and SWE-bench Verified require official post-hoc evaluation and are reported separately; detailed statistics appear in Appendix B.

Aggregate utility does not validate a prescribed organization. Across the five locally scorable benchmarks, complete MAS achieves mean utility 0.4279, exceeding the single-agent and solo-role baselines (Figure 3). Random teams achieve 0.4281. Their paired difference is -0.0002 (stratified task-cluster bootstrap 95% CI $[-0.0308, +0.0311]$), resolving neither an overall advantage nor statistical equivalence. Benchmark effects are heterogeneous: complete MAS is higher on GPQA Diamond, whereas random teams are directionally higher on AIME 2026; the remaining contrasts are smaller. Detailed unadjusted benchmark intervals appear in Appendix B.1. Architecture-level utility is also not monotonic in graph complexity or token use. A7 Graph attains the highest mean utility but consumes approximately 4.5 times as many tokens per run as A1 PEV, whereas A6 Debate reaches similar utility with substantially fewer tokens (Appendix B.3).

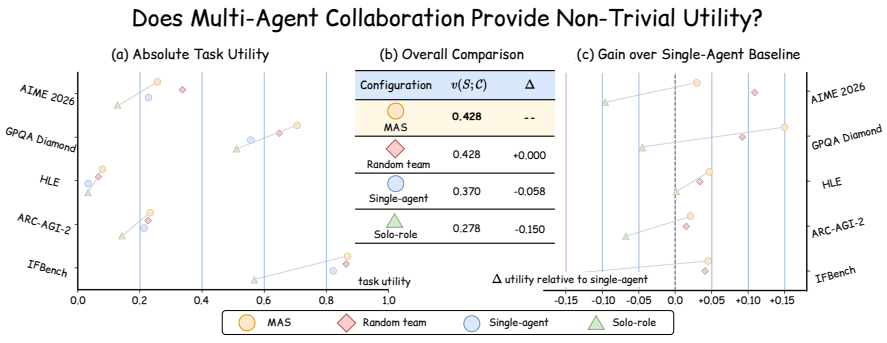


Figure 3: Matched comparison of complete MAS and baseline configurations across five locally scorable benchmarks. Panel (a) reports mean binary task utility for each benchmark. Panel (b) summarizes overall utility; paired uncertainty for $\Delta U = U_{\mathrm{MAS}} - U_{\mathrm{random}}$ is reported in Appendix B.1. Panel (c) reports utility gains relative to the single-agent baseline. Complete MAS denotes the average across architectures A1–A7.

Execution exposes a separate bottleneck. The low scores on difficult benchmarks therefore reflect low-resolution binary task outcomes rather than parsing failures (Appendix B.2). SWE-bench Verified further tests whether a MAS can produce a patch that survives official repository-level execution. Table 1 reports these repository-level outcomes separately from the primary five-benchmark

utility comparison.These results motivate attribution and controlled interventions beyond end-task performance.

Table 1: SWE-bench Verified outcomes for the seven complete MAS architectures. "Error" denotes submitted patches that did not complete under the official evaluation harness.

Architecture	Workload	Execution Status		Task Outcome	
	Submitted	Completed	Error	Resolved	Resolved (%)
A1 PEV	43	2	41	1	2.3
A2 Chain	90	9	81	3	3.3
A3 DAG	87	6	81	**4**	**4.6**
A4 MetaGPT	90	9	81	3	3.3
A5 Star	90	8	82	3	3.3
A6 Debate	89	7	82	3	3.4
A7 Graph	88	8	80	**4**	4.5
Total	**577**	**49**	**528**	**21**	**3.6**

3.3 What Do Different Attribution Methods Measure?

LOO captures full-system fragility, whereas coalition-based methods average marginal utility across contexts. LOO, Shapley, and Banzhaf quantify distinct estimands rather than competing estimates of a single intrinsic property. LOO contributions are sparse: most roles are zero in approximately 87%–93% of runs, with larger effects concentrated in aggregation, supervision, execution, verification, and final-answer positions. Rankings vary across tasks and architectures, and negative LOO occurs when removing a role improves the observed outcome. For each role–architecture–benchmark context, the right panel of Figure 4 reports the fraction of 90 matched LOO comparisons with negative effects. Their variation across contexts shows that negative contribution is local to a realized workflow rather than an intrinsic property of a role. The left panel similarly shows that sampled Shapley and Banzhaf emphasize aggregation, finalization, and verification, while planning, retrieval, and research are often near zero. Candidate-generation roles are therefore not consistently dominant (Appendix C.1).

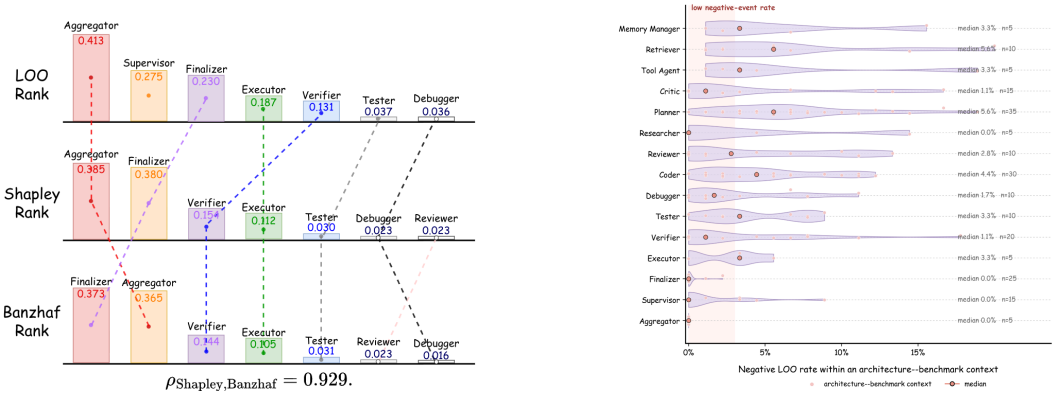


Figure 4: **Attribution estimands and context-dependent negative LOO.** *Left:* leading roles ranked by mean LOO, sampled Shapley, and sampled Banzhaf contribution; dashed paths connect the same role across estimators. Shapley and Banzhaf are strongly rank-correlated, whereas LOO emphasizes dependence within the realized complete workflow. *Right:* distributions of context-level negative LOO rates. Each small point is the fraction of 90 matched comparisons in which removing one role improves utility for a specific architecture–benchmark context. Each cloud collects these rates across contexts for one role, and the outlined marker denotes their median.

Agreement, heterogeneity, and resolution. Sampling-sensitivity analysis supports tier-level rather than exact-rank agreement. Across 50,000 joint resamples of the eight marginal-sample vectors, Shapley and Banzhaf retain the same top-three set in every replicate, although they select the same top-ranked role in only 14.5%. Their cross-estimator Spearman correlation has median 0.934 and 95% range $[0.852, 0.977]$. Exact Shapley calibration in A1 preserves the aggregate role ordering, but the 95th-percentile context–role absolute error is 0.25. Thus, $K = 8$ supports a coarse aggregate

hierarchy, not exact top-rank or per-instance attribution (Appendices C.1 and C.2). Resolution also degrades when binary task outcomes saturate, producing many exactly zero marginals.

3.4 How Do Topology, Role, and Permission Shape Contribution?

Do contribution scores identify where useful reasoning resides? We test three natural interpretations: that contribution is stable for an unchanged role, travels with its functional prompt, and becomes control once a permission is granted. All three fail. Instead, the interventions show that contribution locates *counterfactual fragility* under a particular execution protocol: it can arise from useful reasoning, leverage over an information route, or control of the evaluator-visible output. We probe these mechanisms through validated static routing changes, functional prompt swaps, and runtime permissions while holding tasks, model configuration, and random seeds fixed.

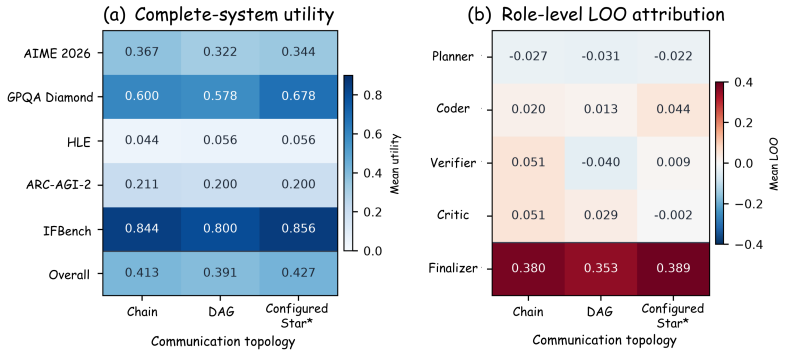


Figure 5: Effects of single-pass routing on team outcomes and local attribution. Chain and DAG realize their configured predecessor sets exactly; the configured-Star condition instead exposes accumulated prior messages and is shown only as an executor diagnostic. Between Chain and DAG, Verifier mean LOO changes from positive to negative, whereas Finalizer remains strongly positive.

Validated rerouting reverses an unchanged position's mean LOO sign. Figure 5 compares fixed roles, prompts, permissions, execution order, and output selection. Runtime audits confirm exact configured predecessor sets for Chain and DAG. Verifier mean LOO changes from $+0.051$ in Chain to -0.040 in DAG, with a paired difference of -0.091 (95% CI $[-0.147, -0.040]$), while team utility changes only modestly. Thus, changing validated static message visibility reverses the sign of an unchanged position's aggregate mean LOO, although task-level effects are sparse and heterogeneous. Finalizer remains strongly positive in both conditions. The configured-Star condition triggers sequential history accumulation and is retained only as an executor diagnostic (Appendix D.1).

Credit can move without utility, and beyond the edited roles. The A5 Planner–Coder swap changes team utility by only -0.002, yet moves their LOO in opposite directions and raises Verifier LOO by $+0.031$ (Figure 6a), redistributing credit without creating utility. Contribution also does not simply follow the exchanged prompt: after the Critic–Finalizer swap, Finalizer LOO rises from 0.387 to 0.427, while the untouched Supervisor increases more sharply from 0.040 to 0.216 (Figure 6b). Moreover, the aggregate effect of $+0.038$ masks opposing task-level outcomes, improving AIME by $+0.200$ but reducing IFBench by -0.144 (Figure 6c). Prompt reassignment can therefore relocate workflow dependence without improving the system and can produce task-specific tradeoffs rather than a universal design rule. Full swap coverage appears in Appendix D.2.

Write access is not work access; authority is not control. Enabling Coder `write_solution` access in A2/A3 changes utility by $+0.018$, while revoking it in A5 changes utility by -0.033. Outcomes remain unchanged in $790/900$ A2/A3 pairs (87.8%) and $316/360$ A5 pairs (87.8%), respectively (Figure 7a). The permission blocks the designated artifact channel rather than ordinary communication, through which task-relevant reasoning can still reach downstream agents. Denying write access therefore need not prevent the Coder from contributing to the solution.

The `final_answer` interventions show when authority becomes a true bottleneck. In A2/A3, excluding the ARC-AGI-2 evaluator-path exception, enabling the sole evaluator-visible path raises mean utility from 0.000 to 0.499 (95% CI $[+0.419, +0.576]$), improving $359/720$ matched pairs (49.9%) and worsening none. In A5, revoking Finalizer authority redirects all 450 runs to Supervisor

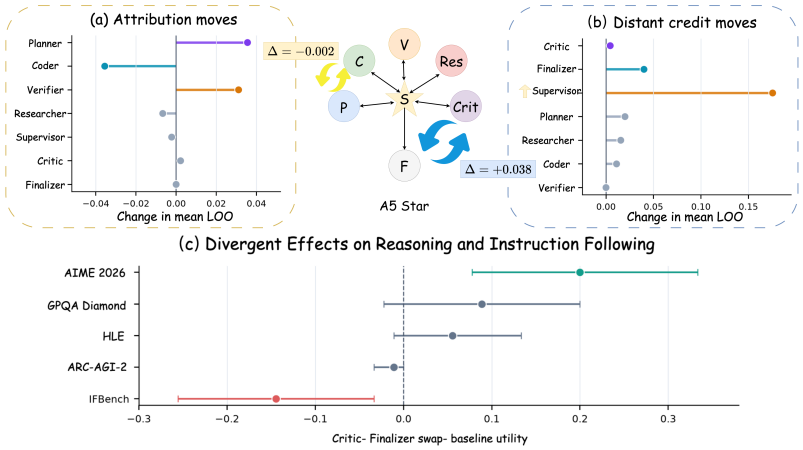


Figure 6: **Role prompts do not carry portable contribution.** (a) Position-level LOO changes for the A5 Planner–Coder swap, despite a near-zero paired team effect. (b) LOO changes for every A5 position after the Critic–Finalizer swap; Supervisor receives no new prompt but exhibits the largest change. (c) Dataset-conditioned team effects for that same swap. Points are paired means and intervals are task-cluster bootstrap 95% intervals.

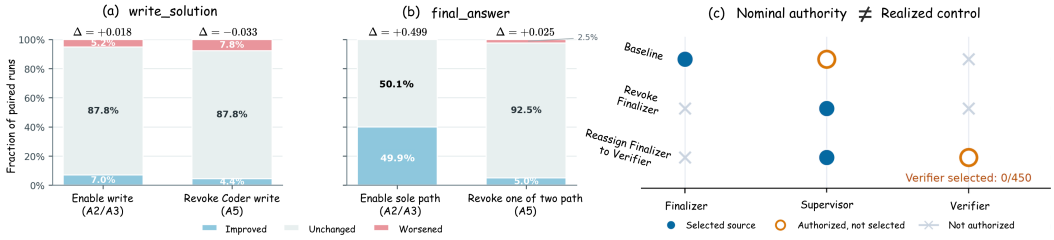


Figure 7: **A permission bit is neither work access nor realized control.** (a–b) Fractions of paired outcomes that improve, remain unchanged, or worsen under `write_solution` and `final_answer` interventions; Δ denotes the mean paired utility effect. The A2/A3 `write_solution` contrast uses all 900 matched pairs, whereas the A2/A3 `final_answer` contrast excludes ARC-AGI-2 and uses 720 matched pairs because its evaluator path does not satisfy the output-validity convention in equation 2. (c) Nominal authority and realized output routing across 450 complete A5 runs per condition. A5 utility effects use 360 valid-evaluator pairs, excluding IFBench.

Table 2: Paired complete-system utility effects for the unified role and permission interventions. Intervals use task-cluster bootstrap resampling. The A2/A3 `final_answer` contrast excludes ARC-AGI-2 and uses 720 matched pairs; the `write_solution` contrast retains all five benchmarks. A5 permission effects use the four benchmarks with valid evaluators.

Mechanism	Contrast and scope	Unchanged	ΔU	95% CI
Role prompt	Planner ↔ Coder, A2/A3/A5	1,175 (87.0%)	+0.008	[−0.016, 0.032]
	Coder ↔ Verifier, A5	391 (86.9%)	+0.007	[−0.024, 0.038]
	Critic ↔ Finalizer, A5	375 (83.3%)	+0.038	[−0.009, 0.084]
Permission, A2/A3	Enable `write_solution`	790 (87.8%)	+0.018	[−0.008, 0.043]
	Enable `final_answer`	361 (50.1%)	+0.499	[+0.419, +0.576]
Permission, A5	Revoke Coder `write_solution`	316 (87.8%)	−0.033	[−0.069, +0.003]
	Assign `write_solution` to Verifier	310 (86.1%)	−0.011	[−0.050, +0.031]
	Revoke Finalizer `final_answer`	333 (92.5%)	+0.025	[−0.006, +0.056]
	Reassign Finalizer authority to Verifier	314 (87.2%)	+0.000	[−0.039, +0.039]

with only a +0.025 utility change, while reassigning the authority to Verifier still yields 0/450 Verifier-selected outputs and zero mean effect (Figure 7b–c). Thus, permission matters only when it controls a unique and actually selected output path.

Taken together, these interventions contradict a context-free reading of position-level contribution. Under null replacement, LOO measures dependence on a position within a specified routing and output-selection protocol. High attribution may reflect useful reasoning, message-path leverage, or control of the evaluator-visible interface; negative attribution may reflect redundancy.

4 Related Work

LLM-based Multi-Agent Systems. LLM-based MAS combine role specialization with structured interaction. Early frameworks established role-based collaboration and programmable conversations (Li et al., 2023; Wu et al., 2023). Subsequent work explored expert recruitment, structured group decision-making, organizational workflows, multi-agent debate, and coordination under partial observability (Chen et al., 2024; Hong et al., 2024; Qian et al., 2024; Du et al., 2023; Agashe et al., 2025). More recent studies investigated scalable communication topologies, automated workflow construction, adaptive orchestration, tool-mediated collaboration, interactive environments, and long-horizon agent tasks (Qian et al., 2025; Zhang et al., 2025a; Dang et al., 2026; Yang et al., 2026a; Gong et al., 2024; Sun et al., 2025; Dong et al., 2024; Toledo et al., 2026). These systems are primarily evaluated by end-task performance, which does not identify the agents or mechanisms responsible for observed gains.

Agent and Component Attribution. Agent- and component-level attribution either quantifies removal effects or allocates utility among coalition members, commonly using Shapley and Banzhaf values (Shapley et al., 1953; Banzhaf III, 1964). Recent work applies these estimators to agentic workflows (Yang et al., 2026b), while AgentSHAP specifically estimates tool importance (Horovicz, 2025). Related studies connect contribution estimation to prompt optimization, cooperative credit assignment, and incentive design (Xia et al., 2026; Hua et al., 2026; Li et al., 2024). *Agents that Matter* highlights sensitivity to coalition sampling, removal semantics, and target metrics (Lu et al., 2026). In contrast, we combine agent-position attribution with matched interventions on topology, role prompts, and runtime permissions to identify the execution mechanisms that shape measured contribution.

Mechanistic and Diagnostic Evaluation. Complementary benchmarks study failure attribution, traces, coordination, role separation, topology, instruction preservation, consensus pollution, and cross-task leakage (Zhang et al., 2025b; Chen et al., 2026; Kim et al., 2026; Mazumder et al., 2026). Broader benchmarks evaluate reasoning, tool use, computer interaction, and long-horizon task completion through system-level outcomes (Liu et al., 2024; Yao et al., 2024; Wang et al., 2024; Xie et al., 2024; Jimenez et al., 2024). These benchmarks provide realistic system-level outcomes but generally do not combine agent-level coalition attribution with matched interventions across topology, functional prompts, and execution permissions.

5 Conclusion

MASContributionBench combines matched baselines, coalition attribution, and controlled interventions across seven architectures and seven task suites. Complete MAS outperform single-agent and solo-role baselines, while their aggregate difference from random teams remains unresolved; collaboration gains alone therefore do not validate a prescribed organization. LOO captures dependence within the complete workflow, whereas sampled Shapley and Banzhaf average marginal utility across coalition contexts.

Our central finding is that measured contribution depends on execution context. Validated Chain–DAG rerouting reverses an unchanged position's mean LOO sign; prompt swaps can redistribute credit, including to unedited roles, without creating utility; and permission produces control only over a unique, selected path. These effects can also reverse across tasks, limiting universal design claims based on aggregate results. High attribution may therefore reflect useful reasoning, routing leverage, or control of the evaluator-visible output. Contribution should be interpreted as counterfactual dependence under a specified execution protocol, not as a direct measure of agent intelligence or capability. Future work should examine multi-round topology execution, alternative removal semantics, and larger coalition-sampling budgets.

AI USE STATEMENT

Generative AI tools were used solely to assist with manuscript writing, translation, language polishing, grammatical correction, and improvements to readability and presentation. They were not used to generate the research ideas, problem formulation, methodology, experimental design, experimental execution, data collection, statistical analysis, result interpretation, or scientific conclusions of this work.

All scientific content and claims were developed and verified by the authors. The authors manually reviewed all AI-assisted text and take full responsibility for the accuracy, originality, and integrity of the manuscript.

ETHICS STATEMENT

This work does not involve human subjects or private personal data. All benchmark datasets are used according to their respective licenses and evaluation protocols. The reported attribution values measure marginal utility under specific tasks, architectures, and execution conditions and should not be interpreted as universal measures of agent intelligence or capability. We additionally report token usage and execution failures to support transparent and responsible comparison of multi-agent systems.

REPRODUCIBILITY STATEMENT

We provide detailed specifications of the experimental protocol to support the reproduction and independent verification of our results. The task suites, task-sampling procedure, model and decoding configurations, execution environment, random seeds, timeout settings, and evaluation protocols are described in Section 3.1. All experiments use the `Qwen/Qwen3.8-27B-FP8` checkpoint (revision `017b9c7af6b5689d5dd426a76e0bc077eb5ca20a`). The main experiments serve the model with vLLM 0.27.1 using two-way tensor parallelism, a 262,144-token context limit, FP8 weights, BF16 computation, and an FP8 KV cache. Requests use temperature 0.2; top-p is not explicitly supplied and defaults to 1.0 under this serving configuration. The default maximum generation length is 2,048 tokens, with documented 4,096-token exceptions for ARC-AGI-2 and designated reasoning roles. Complete serving and dataset-specific decoding settings are reported in Appendix A.1.All direct comparisons use the same task instances, model configuration, and execution seeds. We also report the number of complete-system runs, ablation conditions, coalition evaluations, matched intervention pairs, and execution traces as consistency checks for the experimental outputs.

Appendix A specifies the seven multi-agent architectures, communication topologies, role prompts, execution order, and position-level permissions. Appendix B provides benchmark-level results, scoring outcomes, token usage, and official post-hoc evaluation results. Appendix C documents the implementation and coverage of the LOO, sampled Shapley, and sampled Banzhaf estimators, including the sampling budget and attribution records. Appendix D provides the controlled topology, role-prompt, and runtime-permission interventions, together with their manipulation checks and disaggregated results. The evaluated datasets—AIME 2026, GPQA Diamond, HLE, ARC-AGI-2, IFBench, LiveCodeBench, and SWE-bench Verified—are publicly available under their respective licenses and access conditions.

An anonymized repository containing the implementation, configuration files, evaluation scripts, and reproduction instructions is available at `https://anonymous.4open.science/r/mas-contribution-bench-F976/README.md`.

REFERENCES

Saaket Agashe, Yue Fan, Anthony Reyna, and Xin Eric Wang. Llm-coordination: evaluating and analyzing multi-agent coordination abilities in large language models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 8053–8072, 2025.

John F Banzhaf III. Weighted voting doesn't work: A mathematical analysis. *Rutgers L. Rev.*, 19: 317, 1964.

Mengzhuo Chen, Junjie Wang, Fangwen Mu, Yawen Wang, Zhe Liu, Huanxiang Feng, and Qing Wang. Seeing the whole elephant: A benchmark for failure attribution in llm-based multi-agent systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19888–19905, 2026.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, volume 2024, pp. 20094–20136, 2024.

Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. Arc-agi-2: A new challenge for frontier ai reasoning systems. *arXiv preprint arXiv:2505.11831*, 2025.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench verified. OpenAI, 2024. URL `https://openai.com/index/introducing-swe-bench-verified/`. Originally published August 13, 2024; updated February 24, 2025.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, et al. Multi-agent collaboration via evolving orchestration. *Advances in neural information processing systems*, 38:165025–165059, 2026.

Yubo Dong, Xukun Zhu, Zhengzhe Pan, Linchao Zhu, and Yi Yang. Villageragent: A graph-based multi-agent framework for coordinating complex task dependencies in minecraft. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 16290–16314, 2024.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

Ran Gong, Qiuyuan Huang, Xiaojian Ma, Yusuke Noda, Zane Durante, Zilong Zheng, Demetri Terzopoulos, Li Fei-Fei, Jianfeng Gao, and Hoi Vo. Mindagent: Emergent gaming interaction. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 3154–3183, 2024.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Miriam Horovicz. Agentshap: Interpreting llm agent tool importance with monte carlo shapley value estimation. *arXiv preprint arXiv:2512.12597*, 2025.

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.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Yubin Kim, Chanwoo Park, Taehan Kim, Eugene Park, Samuel Schmidgall, Salman Rahman, Chunjong Park, Cynthia Breazeal, Xin Liu, Hamid Palangi, et al. Teambench: Evaluating agent coordination under enforced role separation. *arXiv preprint arXiv:2605.07073*, 2026.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008, 2023.

Xinran Li, Zifan Liu, Shibo Chen, and Jun Zhang. Individual contributions as intrinsic exploration scaffolds for multi-agent reinforcement learning. *arXiv preprint arXiv:2405.18110*, 2024.

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. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024.

Mingyu Lu, Yushan Huang, Chris Lin, and Su-In Lee. Agents that matter: Optimizing multi-agent llms via removal-based attribution. *arXiv preprint arXiv:2605.27621*, 2026.

Aritra Mazumder, Shubhashis Roy Dipta, Nusrat Jahan Lia, Tanzila Khan, Kainat Raisa Hossain, Nehaa Shri, Shubhrangshu Debsarkar, Humayra Tasnim, Gour Gupal Talukder Shawon, Debjoty Mitra, et al. Agentcollabbench: Diagnosing when good agents make bad collaborators. *arXiv preprint arXiv:2605.08647*, 2026.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hanna Hajishirzi. Generalizing verifiable instruction following. *Advances in Neural Information Processing Systems*, 38, 2026.

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.

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, et al. Scaling large language model-based multi-agent collaboration. In *International Conference on Learning Representations*, volume 2025, pp. 41488–41505, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

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

Haochen Sun, Shuwen Zhang, Lujie Niu, Lei Ren, Hao Xu, Hao Fu, Fangkun Zhao, Caixia Yuan, and Xiaojie Wang. Collab-overcooked: Benchmarking and evaluating large language models as collaborative agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 4922–4951, 2025.

Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Rishi Hazra, Nicolas Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, Minqi Jiang, Alisia Lupidi, et al. Ai research agents for machine learning: Search, exploration, and generalization in mle-bench. *Advances in Neural Information Processing Systems*, 38:35309–35348, 2026.

Jize Wang, Zerun Ma, Yining Li, Songyang Zhang, Cailian Chen, Kai Chen, and Xinyi Le. Gta: a benchmark for general tool agents. *Advances in Neural Information Processing Systems*, 37: 75749–75790, 2024.

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 conversation. *arXiv preprint arXiv:2308.08155*, 2023.

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.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. Agentnet: Decentralized evolutionary coordination for llm-based multi-agent systems. *Advances in Neural Information Processing Systems*, 38:107309–107336, 2026a.

Yingxuan Yang, Bo Huang, Siyuan Qi, Chao Feng, Haoyi Hu, Yuxuan Zhu, Jinbo Hu, Haoran Zhao, Ziyi He, Xiao Liu, et al. Attribution-based analysis and optimization of modular agentic workflows. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 7247–7264, 2026b.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025a.

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*, 2025b.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2026, 2026.

A Implementation Specifications

A.1 Model and Runtime Configuration

All experiments use the `Qwen/Qwen3.8-27B-FP8` checkpoint at revision `017b9c7af6b5689d5dd426a76e0bc077eb5ca20a`. The main inference environment serves the model through an OpenAI-compatible vLLM 0.27.1 backend using two-way tensor parallelism on NVIDIA L20 GPUs. The server uses a maximum context length of $262{,}144$ tokens, FP8 weights, BF16 computation, an `fp8_e4m3` KV cache, and prefix caching. The supplementary H200 runs use the same transferred checkpoint and request-level decoding configuration through two single-GPU vLLM replicas behind a round-robin proxy.

Requests set temperature to 0.2. The client does not explicitly transmit top-p; because the audited L20 service is launched with `--generation-config vllm`, this resolves to the vLLM default of 1.0, rather than the checkpoint's generation default of 0.95. The default maximum output length is $2{,}048$ tokens. ARC-AGI-2 uses a $4{,}096$-token limit with thinking disabled. Beginning with Experiment 2, Coder, Executor, Debugger, Finalizer, Aggregator, and Supervisor use a $4{,}096$-token limit with thinking enabled on AIME 2026, GPQA Diamond, and HLE; other role–dataset combinations use the $2{,}048$-token default with thinking disabled. The default request timeout is 180 seconds, increased to 900 seconds for the long-running supplementary intervention jobs.

A.2 Topology Specifications

Consistent with the main-text configuration $\mathcal{C} = (N, E, \mathbf{r}, \mathbf{p}, \sigma, O, M)$, let $N = \{1, \ldots, n\}$ denote the agent positions. For topology specification, we write the directed design graph as

$$\mathcal{G} = (V, E), \qquad V = N \cup \{o_{\text{out}}\},$$

where $o_{\text{out}} \notin N$ denotes the non-agent output sink (`final_answer`). Each $i \in N$ has a functional prompt r_i and runtime permissions p_i. An edge $(u, v) \in E$ specifies that the output of position u is available to position v; an edge (i, o_{out}) hands the authorized final output to the system-level interface. Thus, o_{out} is part of the design graph but never a coalition member or an attributable agent position.

The topology configurations are defined in YAML files and specify the agent positions, canonical role mappings, entrypoint, terminal nodes, communication edges, orchestration mode, routing policy, aggregation policy, and stopping condition. Table 3 summarizes all seven architectures used in the benchmark.

Table 3: Topology configurations in the benchmark. The listed edges describe the design graph. For cyclic architectures, the current runner executes a bounded single pass rather than repeatedly replaying feedback edges.

ID	Architecture	Roles
A1	Planner–Executor–Verifier	Planner, Executor, Verifier
A2	Chain	Planner, Coder, Reviewer, Finalizer
A3	DAG Workflow	Planner, Retriever, Coder, Debugger, Tester, Finalizer
A4	MetaGPT-lite SOP	Supervisor, Planner, Coder, Tester, Reviewer, Finalizer
A5	Supervisor–Worker Star	Supervisor, Planner, Coder, Verifier, Critic, Researcher, Finalizer
A6	Debate / Committee	Planner, Coder, Debugger, Critic, Verifier, Aggregator
A7	Graph Coordination	Supervisor, Planner, Retriever, Memory Manager, Tool Agent, Coder, Critic, Verifier, Finalizer

A1: Planner–Executor–Verifier. A1 is the smallest role-separated workflow in the benchmark. It first constructs a plan, then executes the main task, and finally verifies the result:

$$E_{\text{A1}} = \{(\texttt{planner}, \texttt{executor}), (\texttt{executor}, \texttt{verifier}), (\texttt{verifier}, o_{\text{out}})\}.$$

The architecture uses sequential execution, fixed-order routing, and verifier-based final aggregation.

A2: Chain. A2 extends the linear workflow with an explicit review stage and a dedicated finalizer:

$$E_{\mathrm{A2}} = \{(\texttt{planner}, \texttt{coder}), (\texttt{coder}, \texttt{reviewer}), (\texttt{reviewer}, \texttt{finalizer}), (\texttt{finalizer}, o_{\mathrm{out}})\}.$$

Its aggregation policy is `finalizer_final`, and the workflow stops when the finalizer returns an answer.

A3: DAG Workflow. A3 introduces parallel context retrieval and implementation followed by debugging, testing, and finalization:

$$E_{\mathrm{A3}} = \{(\texttt{planner}, \texttt{retriever}), (\texttt{planner}, \texttt{coder}), (\texttt{retriever}, \texttt{coder}), (\texttt{coder}, \texttt{debugger}), (\texttt{debugger}, \texttt{tester}), (\texttt{tester}, \texttt{finalizer}), (\texttt{finalizer}, o_{\mathrm{out}})\}.$$

The coder therefore receives information from both the planner and the retriever. This architecture uses dependency-order execution and finalizer-based aggregation.

A4: MetaGPT-lite SOP Workflow. A4 implements a lightweight standard-operating-procedure workflow coordinated by a supervisor:

$$E_{\mathrm{A4}} = \{(\texttt{supervisor}, \texttt{planner}), (\texttt{planner}, \texttt{coder}), (\texttt{coder}, \texttt{tester}), (\texttt{tester}, \texttt{reviewer}), (\texttt{reviewer}, \texttt{finalizer}), (\texttt{finalizer}, \texttt{supervisor}), (\texttt{supervisor}, o_{\mathrm{out}})\}.$$

The return edge from the finalizer to the supervisor is part of the design graph. The architecture uses a supervisor-final aggregation policy and a single bounded execution round.

A5: Supervisor–Worker Star. A5 uses a centralized star-shaped design graph. The supervisor communicates with each specialized worker:

$$E_{\mathrm{A5}} = \{(\texttt{supervisor}, \texttt{planner}), (\texttt{planner}, \texttt{supervisor}), (\texttt{supervisor}, \texttt{researcher}), (\texttt{researcher}, \texttt{supervisor}), (\texttt{supervisor}, \texttt{coder}), (\texttt{coder}, \texttt{supervisor}), (\texttt{supervisor}, \texttt{verifier}), (\texttt{verifier}, \texttt{supervisor}), (\texttt{supervisor}, \texttt{critic}), (\texttt{critic}, \texttt{supervisor}), (\texttt{supervisor}, \texttt{finalizer}), (\texttt{finalizer}, o_{\mathrm{out}})\}.$$

Although the design graph contains bidirectional supervisor–worker edges, the current graph builder does not replay them as additional supervisor turns. Instead, A5 executes as a deterministic single-pass workflow using the `supervisor_mediated_single_pass` routing policy.

A6: Debate / Committee. A6 models a committee-style workflow with candidate generation, critique, verification, and aggregation:

$$E_{\mathrm{A6}} = \{(\texttt{planner}, \texttt{coder}), (\texttt{planner}, \texttt{debugger}), (\texttt{coder}, \texttt{critic}), (\texttt{debugger}, \texttt{critic}), (\texttt{critic}, \texttt{coder}), (\texttt{critic}, \texttt{debugger}), (\texttt{coder}, \texttt{verifier}), (\texttt{debugger}, \texttt{verifier}), (\texttt{critic}, \texttt{verifier}), (\texttt{verifier}, \texttt{aggregator}), (\texttt{aggregator}, o_{\mathrm{out}})\}.$$

The cyclic edges between the critic and candidate-producing positions describe the intended debate structure. However, the current implementation executes the graph once in a deterministic order and does not iterate additional debate rounds. The aggregator is the designated final-answer position.

A7: Graph Coordination. A7 is the richest coordination graph in the benchmark. It combines supervision, planning, retrieval, memory management, tool use, coding, critique, verification, and finalization:

$$
\begin{aligned}
E_{\mathrm{A7}} = \{&(\texttt{supervisor}, \texttt{planner}), (\texttt{supervisor}, \texttt{retriever}), (\texttt{supervisor}, \texttt{memory_manager}),\\
&(\texttt{planner}, \texttt{retriever}), (\texttt{retriever}, \texttt{memory_manager}), (\texttt{memory_manager}, \texttt{planner}),\\
&(\texttt{planner}, \texttt{coder}), (\texttt{retriever}, \texttt{coder}),\\
&(\texttt{coder}, \texttt{tool_agent}), (\texttt{tool_agent}, \texttt{coder}),\\
&(\texttt{coder}, \texttt{critic}), (\texttt{critic}, \texttt{coder}),\\
&(\texttt{coder}, \texttt{verifier}), (\texttt{tool_agent}, \texttt{verifier}),\\
&(\texttt{critic}, \texttt{verifier}), (\texttt{verifier}, \texttt{finalizer}),\\
&(\texttt{memory_manager}, \texttt{finalizer}), (\texttt{finalizer}, \texttt{supervisor}),\\
&(\texttt{supervisor}, o_{\mathrm{out}})\}.
\end{aligned}
$$

A7 uses graph-based single-pass routing. Its cycles and feedback edges represent the communication design, but the current runner does not perform iterative graph-state rounds.

Role execution and message availability. For each agent position $i \in N$, the runtime constructs its context from already executed in-neighbors:

$$\mathrm{Upstream}_{\sigma}(i) = \{j \in N \mid (j, i) \in E,\ \sigma(j) < \sigma(i)\}.$$

Thus, position i receives the structured outputs of the positions in $\mathrm{Upstream}_{\sigma}(i)$, together with a topology context containing its canonical role label, architecture identifier, architecture family, and edge list. This is the same visibility condition used in equation 1; outputs are forwarded only along configured edges.

For acyclic role graphs, the execution order is obtained by topological sorting. For cyclic architectures, including A4–A7, the runner uses a deterministic single-pass fallback order. In debate, cyclic, and round-robin modes, finalizer-like positions are moved to the end of the declared role list. Thus, the edges of A5–A7 specify the intended communication structure but do not imply repeated message-passing rounds.

Agent removal. The implementation supports three removal-related settings. Under hard removal, the target position $i \in N$ is skipped and produces no output. Under null-agent replacement, the position remains in the execution graph but returns a fixed structured null output:

```
{
  "summary": "null agent replacement",
  "artifact": "",
  "evidence": [],
  "confidence": "low",
  "failure_modes": []
}
```

Under bypass removal of position i, the position is deleted and each incoming predecessor is connected to each outgoing successor:

$$E^{\mathrm{bypass}(i)} = (E \setminus \{(u, v) \in E : u = i \vee v = i\}) \cup \{(u, v) : (u, i) \in E,\ (i, v) \in E\}.$$

These settings distinguish the effect of missing content from the effect of rewiring the communication structure.

All LOO, sampled Shapley, and sampled Banzhaf results reported in the paper use null-agent replacement: inactive positions remain in the original execution graph but return the fixed structured null output. Thus, the methods differ in the coalition contexts and weighting schemes they evaluate, not in how an absent role is implemented.

A.3 Role Prompt Specifications

Each agent in the benchmark is assigned a role-specific prompt that defines its responsibilities, behavioral boundaries, and output format. The prompt determines what an agent is expected to

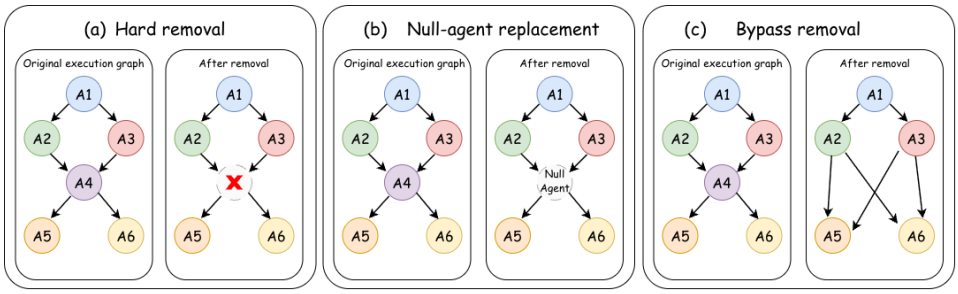


Figure 8: **Agent-removal protocols.** (a) *Hard removal* skips the target position, which produces no output. (b) *Null-agent replacement* preserves the target position, its incident edges, and the execution order, but replaces its substantive output with a fixed structured null response. (c) *Bypass removal* deletes the target position and directly connects each of its predecessors to each of its successors.

do, whereas the topology determines which messages and intermediate outputs are available to that agent.

The benchmark uses canonical prompts for semantically equivalent roles. In particular, Executor uses the Coder prompt, Reviewer uses the Critic prompt, Retriever uses the Researcher prompt, Tester uses the Verifier prompt, and Aggregator uses the Finalizer prompt. This design separates functional role from architecture-specific role naming.

Table 4: Role-to-prompt mapping in the benchmark implementation.

Role	Canonical prompt	Main responsibility
Planner	Planner	Task decomposition and execution planning
Coder	Coder	Main artifact or solution generation
Executor	Coder	Execution of the planned solution
Debugger	Debugger	Diagnosis and targeted repair
Verifier	Verifier	Correctness and constraint checking
Tester	Verifier	Testing and validation
Critic	Critic	Identification of weaknesses and failure modes
Reviewer	Critic	Review of the current artifact
Researcher	Researcher	Collection and organization of contextual information
Retriever	Researcher	Retrieval of contextual information
Supervisor	Supervisor	Coordination and information routing
Finalizer	Finalizer	Synthesis of intermediate outputs
Aggregator	Finalizer	Aggregation of candidate outputs
Memory Manager	Memory Manager	Maintenance of task-relevant state
Tool Agent	Tool Agent	Tool-mediated execution and inspection

Common output contract. All role prompts require the agent to return a JSON object with the following fields:

```
{
  "summary": "one or two sentences describing what you did",
  "artifact": "your role-specific output",
  "evidence": ["short evidence items, constraints, tests, or references used"],
  "confidence": "low | medium | high",
  "failure_modes": ["possible issues or empty list"]
}
```

The agent must not include Markdown outside the JSON object or expose private deliberation. This common schema makes intermediate outputs machine-readable and separates the main artifact from supporting evidence, confidence, and unresolved failure modes.

Planner prompt. The Planner is responsible for understanding the task, identifying constraints, decomposing the work, and producing a clear execution plan for downstream agents. It should re-

state the task in operational terms, identify required inputs, edge cases, and evaluation criteria, break the task into ordered steps, assign responsibilities to downstream roles, and flag missing information or risky assumptions.

The Planner does not implement the final solution unless explicitly instructed by the orchestration policy. It must not fabricate tests or results, claim correctness without verifier evidence, or produce the final answer without final authority.

Coder prompt. The Coder is responsible for producing the main task artifact, which may be code, a patch, a structured answer, or an executable solution. It should follow the planner's constraints when available while prioritizing the original task specification in case of conflict.

The Coder should implement a concrete solution, keep it minimal and aligned with the required task format, preserve required function names, entry points, file names, and output formats, and state assumptions that affect correctness. It must prepare the artifact for later verification and must not invent successful test results or ignore evaluator constraints.

Debugger prompt. The Debugger revises a previous solution after critique, test failure, runtime error, or suspected defect. It focuses on diagnosis and targeted correction rather than rewriting the entire artifact.

The Debugger should identify the likely cause of failure, propose a minimal fix, preserve valid parts of the existing solution, explain which failure mode the revision addresses, and maintain compatibility with the task evaluator and output format. It must not make broad unrelated refactors, claim that tests passed without evidence, or change public APIs and required entry points unless required by the task.

Verifier prompt. The Verifier checks whether an artifact satisfies the task specification and evaluation criteria. It may reason about tests, constraints, edge cases, and expected behavior.

The Verifier should compare the artifact with the original task, identify correctness issues and missing requirements, report whether the artifact appears passable under the evaluator, provide actionable verification evidence, and recommend or perform specific checks when permitted by the assigned permissions.

When the architecture uses the Verifier as the final-producing role, it returns the best verified task answer rather than a verification report. Unless the architecture requires it to provide the final answer, the Verifier should not rewrite the complete solution. It must distinguish verified evidence from plausible assumptions and must not approve an artifact with missing outputs, tests, or required constraints.

Critic prompt. The Critic challenges the current plan or artifact in order to identify weaknesses and improve robustness. It is not primarily responsible for final synthesis.

The Critic should identify ambiguous assumptions, missing constraints, fragile logic, overfitting to examples, ignored edge cases, and relevant alternatives. It should produce concise and actionable criticism for the Coder, Debugger, or Finalizer and indicate whether an issue is severe, moderate, or minor when appropriate.

The Critic must not reject an artifact without a concrete reason, produce the final answer without explicit authority, or add unrelated preferences that are not connected to the task or evaluator.

Researcher prompt. The Researcher collects and organizes relevant contextual information. This may include facts, specifications, constraints, examples, or other information needed by downstream agents.

The Researcher should extract relevant facts and constraints from the task context, summarize external or long-context information, identify missing context that may affect correctness, provide citations or source labels when available, and separate retrieved information from interpretation.

The Researcher does not implement the main solution unless explicitly asked. It must not invent references, files, APIs, or test results, and should avoid adding unsupported facts.

Supervisor prompt. The Supervisor coordinates the multi-agent workflow. It routes information, decides which role should act next, resolves conflicts between agents, and may produce or approve the final answer when the architecture assigns it final authority.

The Supervisor should track the current task state and each agent's contribution, route work to appropriate roles, resolve contradictions between planner, coder, verifier, critic, and researcher outputs, decide when enough evidence exists to finalize, and preserve role boundaries.

The Supervisor must not hide unresolved verifier or critic concerns, overwrite specialist evidence without explanation, or claim tool or test results that are not present in the execution trace. When it is the last or final-authority node, it should output the final task answer rather than routing instructions.

Finalizer prompt. The Finalizer synthesizes intermediate artifacts into the final answer required by the task. It should preserve validated content and remove internal process discussion.

The Finalizer must produce the final answer in the required format, incorporate verified fixes and relevant critiques, respect required file names, entry points, schemas, and answer styles, and omit unnecessary process chatter. It should state unresolved risks only when the final answer format allows it.

The Finalizer must not introduce unsupported functionality at the last stage, ignore failed verification evidence, or include multiple competing answers unless explicitly requested.

Memory Manager prompt. The Memory Manager maintains task-relevant state across the workflow. It identifies what information should be remembered, retrieved, updated, or discarded so that downstream agents can use consistent context.

The Memory Manager should summarize durable task state, constraints, decisions, and open issues; retrieve relevant prior state when requested; detect contradictions between current messages and stored state; mark stale or superseded information; and keep memory concise and structured.

The Memory Manager must not act as the main Coder, Verifier, or Finalizer, store unsupported claims as facts, or expose private or irrelevant information.

Tool Agent prompt. The Tool Agent performs tool-mediated operations when permitted by the permission configuration. These operations may include running tests, executing code, inspecting files, formatting outputs, or returning structured tool results to other agents.

The Tool Agent should execute only the requested tool action, return exact results, errors, and relevant metadata, distinguish successful execution from failure, avoid changing task artifacts unless explicitly asked, and keep outputs structured for the Verifier, Coder, or Supervisor.

The Tool Agent must not fabricate tool outputs, make final judgments beyond the available evidence, or perform actions outside the granted permissions.

B Supplementary Results for the Attribution Setting

This section reports result slices omitted from the main text: per-benchmark baseline scores, scoring outcomes, architecture-level cost, official LiveCodeBench results, and baseline failure composition. The primary local evaluation contains 30 fixed tasks, three seeds, and 630 complete-MAS runs per benchmark. SWE-bench Verified is not repeated because its full harness statistics already appear in Table 1.

B.1 Benchmark-level Comparison

For the MAS–random comparison, we first average the seven complete architectures and the three random-team sample identifiers within each task–seed pair. This yields 450 paired observations from 150 task clusters, each retaining three execution seeds. Overall uncertainty uses 20,000 benchmark-stratified task-cluster bootstrap replicates with seed 20260923; benchmark-specific intervals resample tasks within the corresponding suite. All intervals are percentile 95% intervals.

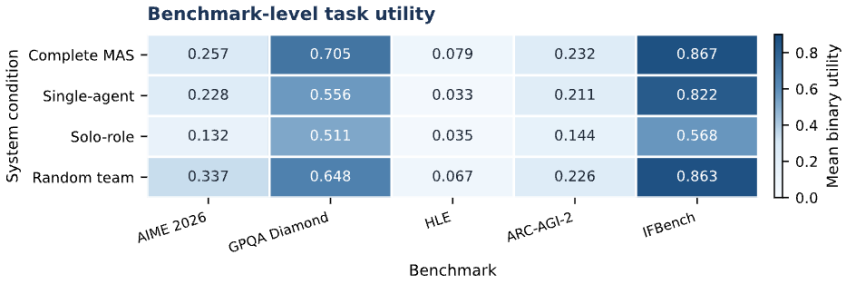


Figure 9: Mean binary utility on the five locally scorable benchmarks. Complete MAS averages architectures A1–A7; baseline rows use the matched conditions defined in Section 3.1.

The largest complete-MAS advantage over random teams occurs on GPQA Diamond ($+0.057$), whereas AIME 2026 reverses the ordering (-0.080). The heatmap therefore exposes benchmark heterogeneity hidden by the equal overall means in the main text.

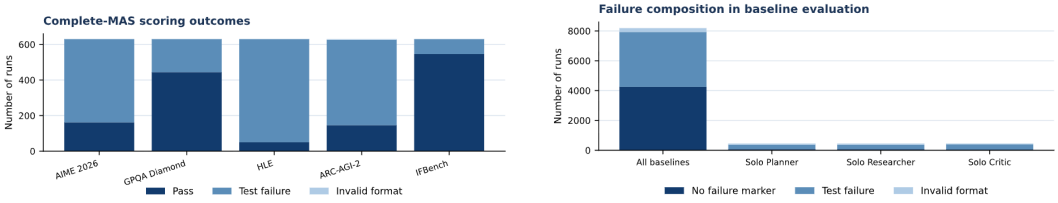
Table 5: Paired complete-MAS versus random-team utility. Differences are defined as $\Delta U = U_{\text{MAS}} - U_{\text{random}}$. Intervals use 20,000 task-cluster bootstrap replicates; the overall interval is stratified by benchmark. Benchmark-level intervals are unadjusted exploratory contrasts.

Scope	MAS	Random	ΔU	95% CI
Overall	0.4279	0.4281	-0.0002	$[-0.0308, +0.0311]$
AIME 2026	0.2571	0.3370	-0.0799	$[-0.1725, +0.0143]$
GPQA Diamond	0.7048	0.6481	$+0.0566$	$[+0.0026, +0.1307]$
HLE	0.0794	0.0667	$+0.0127$	$[-0.0598, +0.0852]$
ARC-AGI-2	0.2317	0.2259	$+0.0058$	$[-0.0513, +0.0598]$
IFBench	0.8667	0.8630	$+0.0037$	$[-0.0439, +0.0540]$

The overall interval permits effects of approximately three percentage points in either direction and therefore establishes neither superiority nor equivalence. GPQA Diamond is the only benchmark whose nominal interval excludes zero, favoring complete MAS by 0.0566. Because the five benchmark contrasts are unadjusted and exploratory, this result should not be interpreted as a general architecture advantage. AIME shows a larger point estimate in the opposite direction, while its interval includes zero.

B.2 Task Failure and Output Validity

Figure 10a summarizes the unmodified complete-MAS reference runs, for which correct, valid-but-incorrect, and invalid-format outcomes have the same interpretation across architectures. Across 3,150 runs, 1,348 are correct, 1,798 are valid but incorrect, and only four are invalidly formatted. Thus, low utility on the difficult suites primarily reflects task failure rather than parsing failure.



(a) Complete-MAS scoring outcomes.

(b) Failure composition in matched conditions.

Figure 10: **Analysis of execution outcomes and failure compositions.** Left: Complete-MAS scoring outcomes across 3,150 runs (7 architectures, 30 tasks, and 3 execution seeds). Right: Failure composition in matched baseline conditions separating task failure, invalid format, and runs without a failure marker.

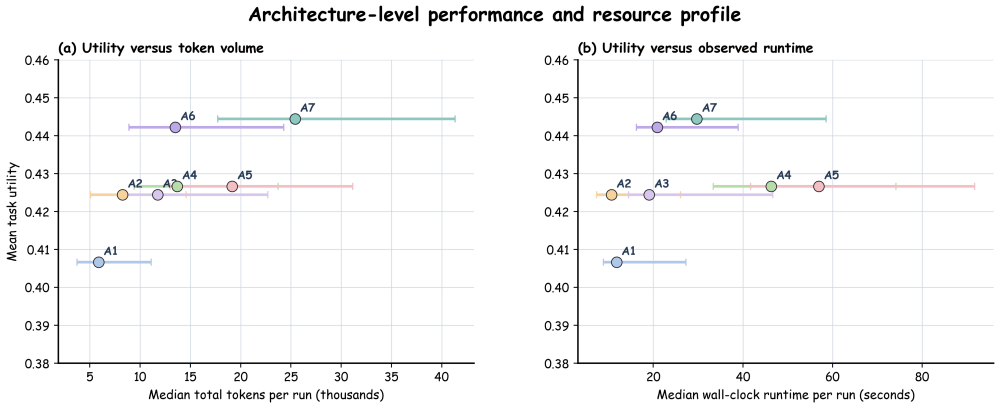


Figure 11: Architecture-level utility and resource use across 450 complete-MAS runs per architecture. Each point shows mean task utility against median total tokens (left) or median observed wall-clock runtime (right); horizontal bars show interquartile ranges. A6 Debate attains utility close to A7 Graph while using fewer tokens and less observed runtime.

The matched baselines exhibit a different failure profile (Figure 10b). Most unsuccessful runs remain task failures, but invalid formats concentrate in standalone Planner, Researcher, and Critic conditions. Their solo scores therefore reflect end-to-end submission ability, rather than the value of their intermediate outputs when downstream roles consume them. These baseline results should not be compared directly with the complete-MAS format rate because the evaluated output protocols differ.

B.3 Architecture-level Performance and Resource Use

We report resource use in terms of generated and input tokens together with observed end-to-end wall-clock runtime. The local execution records do not contain provider-priced USD costs; token counts therefore serve as the model-use proxy. Runtime includes local execution and caching effects and is reported descriptively rather than as hardware-independent latency.

Table 6 reports the corresponding numerical summary. A7 Graph has the highest observed mean utility (0.444), although A6 Debate is descriptively similar (0.442) while using a median of 13.5k rather than 25.4k total tokens and taking 20.9 rather than 29.7 seconds of observed runtime. These descriptive results show no monotonic relationship between graph complexity, resource use, and utility. Runtime comparisons should be interpreted cautiously because cache-hit rates differ across conditions.

Table 6: Architecture-level resource profile. Token and wall-clock values are medians with IQR across 450 complete-MAS runs per architecture. Bold denotes the highest observed mean utility; underlining denotes the lowest value in a resource column. USD cost is unavailable because all runs use a local Qwen/vLLM backend and no provider price is logged.

Architecture	Utility	Tokens	Runtime (s)	Input/output	Cache hit rate
A1	0.407	5.9k [3.7, 11.1]	11.8 [8.9, 27.3]	4.0k / 1.1k	54%
A2	0.424	8.3k [5.1, 14.6]	10.6 [7.4, 26.1]	5.5k / 1.5k	68%
A3	0.424	11.7k [8.2, 22.7]	19.1 [14.4, 46.7]	8.6k / 2.1k	56%
A4	0.427	13.7k [9.4, 23.7]	46.3 [33.4, 74.1]	10.3k / 2.2k	37%
A5	0.427	19.2k [13.0, 31.2]	56.9 [41.7, 91.7]	15.9k / 2.5k	34%
A6	0.442	13.5k [8.9, 24.3]	20.9 [16.2, 38.9]	10.2k / 2.1k	64%
A7	**0.444**	25.4k [17.7, 41.3]	29.7 [22.9, 58.5]	20.8k / 3.2k	60%

Table 7 and Figure 12 show substantial benchmark heterogeneity. ARC-AGI-2 has the largest median token use (30.9k) and runtime (101.0 seconds) but low mean utility (0.232), whereas IFBench reaches the highest mean utility (0.867) with substantially lower median token use (7.7k). Thus,

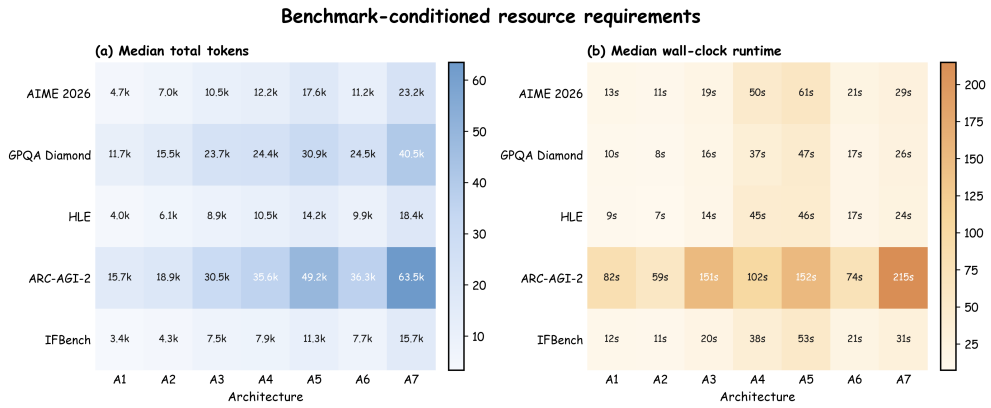


Figure 12: Benchmark- and architecture-level resource use. Each cell summarizes 90 runs (30 task instances and three seeds) by its median total token count or observed wall-clock runtime. Resource demand depends strongly on the task suite as well as on architecture.

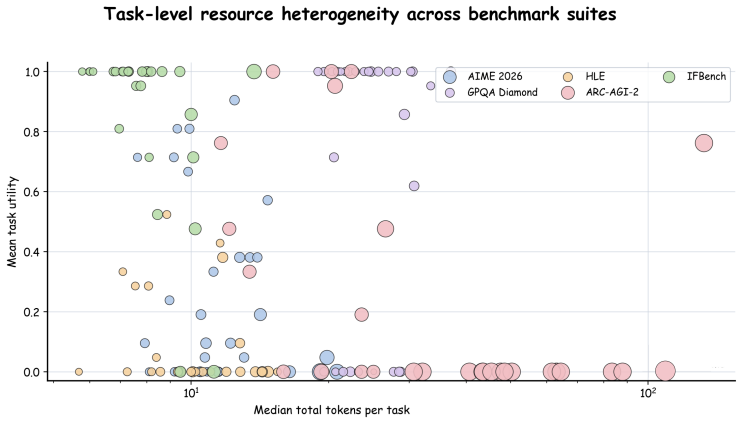


Figure 13: Task-level utility and resource use. Each point aggregates 21 runs: seven architectures and three seeds. The horizontal axis shows median total tokens, color denotes benchmark, and point area encodes median observed wall-clock runtime.

aggregate resource use should not be interpreted as a direct proxy for task difficulty or system effectiveness.

Table 7: Benchmark-level resource profile across all seven architectures. Values are medians with IQR over 630 complete-MAS runs per benchmark. Bold denotes the highest utility; underlining denotes the lowest resource use.

Benchmark	Utility	Tokens	Runtime (s)
AIME 2026	0.257	11.6k [8.2, 18.0]	25.1 [15.3, 53.3]
GPQA Diamond	0.705	24.1k [18.2, 31.4]	17.2 [11.6, 31.8]
HLE	0.079	9.9k [7.0, 14.4]	17.7 [11.2, 35.9]
ARC-AGI-2	0.232	30.9k [19.8, 55.4]	101.0 [59.0, 195.3]
IFBench	**0.867**	7.7k [5.0, 11.0]	23.0 [13.5, 39.5]

Task-level heterogeneity. Aggregating over architectures and seeds yields 150 task-level observations. Across these tasks, total tokens and mean utility have a modest negative Spearman correlation ($\rho = -0.228$), and observed runtime and mean utility show a similarly negative association ($\rho = -0.238$). These descriptive patterns reinforce that additional computation does not reliably translate into higher binary task utility.

Table 8: LiveCodeBench code-generation scores from the official evaluator. Each seed evaluates all 701 tasks. Bold denotes the largest observed mean across three execution seeds; range is the difference between the largest and smallest seed-level scores. These summaries are descriptive and do not constitute uncertainty intervals or statistical rankings.

Architecture	Seed 0	Seed 1	Seed 2	Mean	Range
A1	0.0713	0.0799	0.0699	0.0737	0.0100
A2	0.0799	0.0799	0.0799	0.0799	0.0000
A3	0.0813	0.0856	0.0813	0.0827	0.0043
A4	0.0785	0.0942	0.0799	**0.0842**	0.0157
A5	0.0799	0.0884	0.0799	0.0827	0.0086
A6	0.0770	0.0913	0.0770	0.0818	0.0143
A7	0.0799	0.0884	0.0827	0.0837	0.0086

B.4 LiveCodeBench Official Evaluation

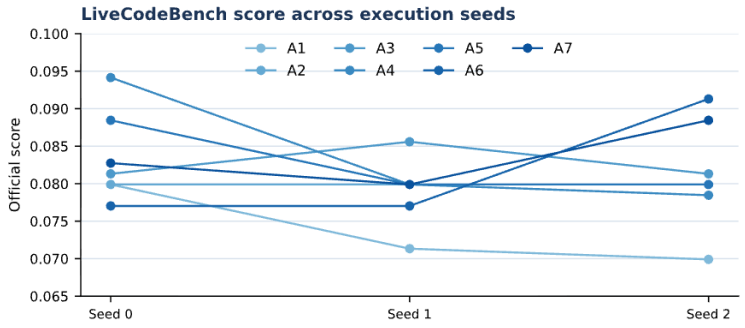


Figure 14: Official LiveCodeBench scores across three execution seeds. Each point evaluates one complete architecture on the full 701-task set.

LiveCodeBench uses all 701 tasks for each complete architecture and seed, yielding 21 official evaluation groups. It remains separate from the primary comparison because the baselines use a 30-task subset.

All 21 LiveCodeBench evaluations completed successfully under the official code-generation evaluator, with 701 tasks evaluated for every architecture–seed pair. A4 MetaGPT-lite obtains the highest mean score (0.0842), followed closely by A7 Graph (0.0837). However, the ranking is not seed-invariant: A3 is highest for seed 0, A4 for seed 1, and A7 for seed 2. In particular, A4 has the largest seed range (0.0157), so its mean lead should be interpreted descriptively rather than as evidence of a stable architecture advantage. These official results remain separate from the primary matched five-benchmark comparison because they use a different task collection and evaluation protocol.

C Detailed Attribution Results

This section reports attribution coverage, complete global role summaries, estimator agreement, and benchmark- and architecture-conditioned results. The diagnostics are necessary because binary task utility produces many exactly zero marginal effects and because roles do not occur in the same set of architectures.

C.1 Global Attribution Profiles and Estimator Agreement

Table 9 reports every role present in either attribution experiment. Intermediate LOO positions are exactly zero in roughly 87%–93% of runs, but aggregation and output positions are much less sparse: Aggregator is zero in 58.7% of LOO runs and Finalizer in 76.7%. Thus, the earlier sparsity pattern is not uniform across the graph. Negative means for Critic, Tool Agent, Retriever, and Memory Manager indicate that removal improves the observed binary outcome on average in their evaluated contexts; they do not establish globally harmful capabilities.

Table 9: Complete global role summaries. "Neg./zero" gives the percentage of negative and exactly zero records. Experiment 3 LOO covers seven architectures; Experiment 4 coalition attribution covers A1, A2, A3, A5, and A6, so columns across experiments are descriptive rather than paired.

	LOO			sampled coalitions			
Role	n	**Mean**	**Neg./zero (%)**	n	**Shapley**	**Banzhaf**	**Shapley neg./zero (%)**
Aggregator	450	0.413	0.0 / 58.7	450	0.385	0.365	1.6 / 46.0
Supervisor	1,350	0.275	1.4 / 69.7	450	0.022	0.011	10.4 / 74.0
Finalizer	2,250	0.230	0.1 / 76.7	1,350	0.380	0.373	0.7 / 47.6
Executor	450	0.187	2.4 / 76.4	450	0.112	0.105	2.0 / 70.7
Verifier	1,800	0.131	3.4 / 80.2	1,350	0.154	0.144	5.0 / 65.2
Tester	900	0.037	4.3 / 87.7	450	0.030	0.031	7.8 / 76.4
Debugger	900	0.036	3.4 / 89.6	900	0.023	0.016	4.9 / 81.1
Coder	2,700	0.019	4.8 / 88.4	1,800	0.019	0.014	6.3 / 81.3
Reviewer	900	0.019	4.8 / 88.6	450	0.023	0.023	8.4 / 77.8
Researcher	450	0.018	3.8 / 90.7	450	0.000	0.003	9.8 / 80.9
Planner	3,150	0.001	6.5 / 87.0	2,250	0.00006	0.001	7.8 / 83.6
Critic	1,350	−0.004	3.9 / 92.6	900	0.005	−0.001	6.1 / 85.1
Tool Agent	450	−0.009	6.0 / 88.9	–	–	–	–
Retriever	900	−0.022	6.4 / 89.3	450	0.0003	0.001	5.6 / 88.0
Memory Manager	450	−0.022	5.8 / 90.7	–	–	–	–

The global cross-experiment ranks should not be read as a direct estimator contest. For example, Supervisor has mean LOO 0.275 because Experiment 3 also includes the A4 and A7 positions, but its Experiment 4 coalition mean is computed only in A5. Conversely, Experiment 4 evaluates Finalizer only in the architectures where it is the terminal output position. Within Experiment 4, where Shapley and Banzhaf share identical coverage, their rankings are highly consistent (Table 10). The weakest agreement occurs on IFBench ($\rho = 0.786$) and in A2 Chain (0.800), while the high-level ordering of output, verification, and intermediate positions remains stable.

Table 10: Complete Shapley–Banzhaf rank agreement.

Global or benchmark context	ρ	**Architecture context**	ρ
Global roles	0.929	A1 PEV	1.000
AIME 2026	0.956	A2 Chain	0.800
GPQA Diamond	0.907	A3 DAG	1.000
HLE	0.962	A5 Star	0.857
ARC-AGI-2	0.960	A6 Debate	1.000
IFBench	0.786	–	–

Coalition-sampling sensitivity. Each of the 23,400 attribution records stores its eight original marginal values. To isolate sensitivity to coalition or permutation sampling, we align the same sample index across tasks, seeds, and architectures, aggregate these records into eight global role vectors, and jointly resample the eight vectors 50,000 times. This procedure conditions on the observed tasks and model runs; its intervals do not include task-sampling or execution uncertainty.

Table 11: Global ranking sensitivity to the eight coalition or permutation samples. Ranges are the central 95% of 50,000 joint bootstrap replicates.

Diagnostic	**Sampled Shapley**	**Sampled Banzhaf**
Observed top-1	Aggregator	Finalizer
Observed top-1 retained	86.8%	98.2%
Alternative top-1	Finalizer (13.2%)	Aggregator (1.8%)
Top-three set retained	100%	100%
Spearman median	0.978	0.989
Spearman 95% range	$[0.934, 1.000]$	$[0.934, 1.000]$
Kendall median	0.923	0.949
Kendall 95% range	$[0.821, 1.000]$	$[0.846, 1.000]$

The two estimators have observed global Spearman correlation 0.929 and Kendall correlation 0.769. Under joint resampling, their cross-estimator Spearman correlation has median 0.934 (95% range

$[0.852, 0.977]$), and Kendall correlation has median 0.813 ($[0.692, 0.897]$). Their top-three sets agree in every replicate, but their top-ranked roles agree in only 14.5%, because Aggregator and Finalizer frequently exchange first place. Agreement is therefore stable at the tier level rather than at the exact-rank level.

Table 12: Sampling-sensitivity intervals for leading aggregate roles. These intervals condition on the observed tasks and executions.

Method	Role	Estimate	95% interval
Shapley	Aggregator	0.385	$[0.379, 0.392]$
Shapley	Finalizer	0.380	$[0.378, 0.383]$
Shapley	Verifier	0.154	$[0.149, 0.159]$
Shapley	Executor	0.112	$[0.108, 0.116]$
Banzhaf	Finalizer	0.373	$[0.370, 0.377]$
Banzhaf	Aggregator	0.365	$[0.359, 0.371]$
Banzhaf	Verifier	0.144	$[0.141, 0.146]$
Banzhaf	Executor	0.105	$[0.097, 0.113]$

These leading roles remain positive in every bootstrap replicate. In contrast, near-zero roles have unstable signs: Shapley Planner and Retriever are positive in 50.1% and 54.4% of replicates, respectively, while Researcher is approximately balanced across signs. Banzhaf Planner and Retriever are positive in 69.0% and 64.4%, and Critic is negative in 62.1%. Accordingly, we do not interpret the signs of near-zero roles as stable.

Exact-Shapley calibration in A1. A1 has three positions, and all 450 task–seed contexts contain all $2^3 = 8$ coalitions without missing or conflicting records. This permits exact Shapley computation without additional model executions.

Table 13: Exact and $K = 8$ sampled Shapley values in A1. MAE is the mean context-level absolute error for each role.

Role	Exact	Sampled	Bias	MAE
Planner	0.0056	0.0031	-0.0025	0.0212
Executor	0.1122	0.1122	0.0000	0.0469
Verifier	0.3911	0.3936	$+0.0025$	0.0442

Exact and sampled values give the same aggregate ordering, Verifier $>$ Executor $>$ Planner. The sampled and exact top-1 sets overlap in all 450 contexts, while the complete tie sets agree in 76.7%. Context–role absolute error has median zero but 95th percentile 0.25. Thus, $K = 8$ accurately recovers the aggregate A1 ordering while remaining too noisy for fine-grained attribution on an individual task execution.

C.2 Benchmark- and Architecture-conditioned Attribution

Figure 15 reports the three leading sampled Shapley positions in every evaluated benchmark and architecture. The leader is not a fixed role name. Aggregator and Finalizer dominate GPQA Diamond and IFBench; Executor becomes competitive on AIME and leads ARC-AGI-2; HLE gives the same small value (0.046) to its three leading positions. Across architectures, the dominant value follows the evaluator-visible output position: Verifier in A1, Finalizer in A2/A3/A5, and Aggregator in A6.

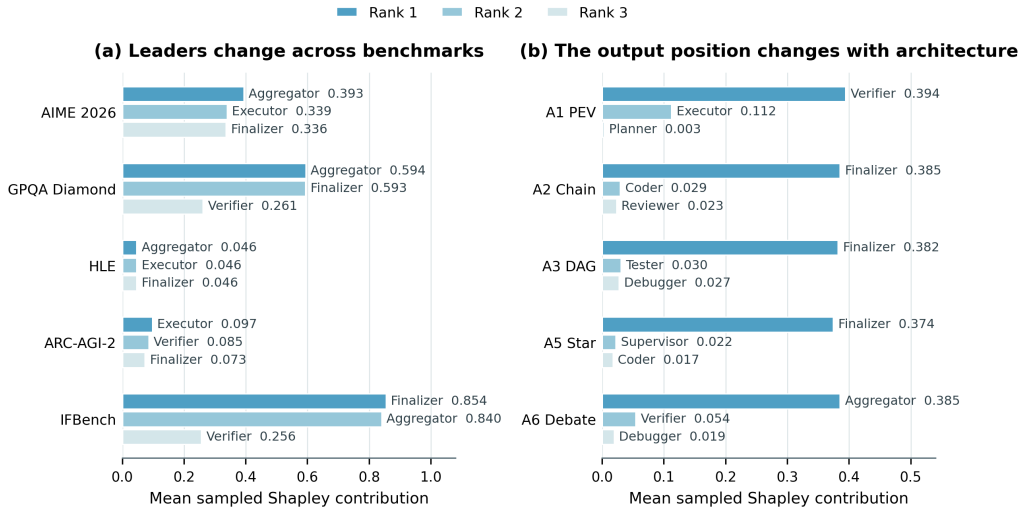


Figure 15: Top-three benchmark- and architecture-conditioned sampled Shapley positions. Bars show means across 30 tasks, three execution seeds, and eight sampled permutations per role. Colors encode within-context rank rather than role identity.

Table 14 separates contribution magnitude from attribution resolution. HLE has the smallest mean Shapley value and 93.3% exact zeros, consistent with failure saturation. IFBench has the largest mean Shapley value but the lowest cross-estimator rank agreement, consistent with success saturation, output-path dominance, and finite sampling. Architecture-level team utility varies only from 0.436 to 0.509, whereas mean attribution and sparsity still vary substantially; these statistics therefore describe how observable marginal value is distributed, not which architecture is universally best.

Table 14: Conditioned utility and sampled-Shapley identifiability diagnostics.

Context	Level	Full-team U	Mean Shapley	Negative (%)	Zero (%)
AIME 2026	Benchmark	0.491	0.094	13.0	52.6
GPQA Diamond	Benchmark	0.664	0.128	3.7	76.2
HLE	Benchmark	0.069	0.013	1.7	93.3
ARC-AGI-2	Benchmark	0.233	0.045	4.1	79.5
IFBench	Benchmark	0.851	0.164	6.2	69.1
A1 PEV	Architecture	0.509	0.170	3.0	66.7
A2 Chain	Architecture	0.438	0.109	5.6	73.2
A3 DAG	Architecture	0.456	0.076	5.0	77.2
A5 Star	Architecture	0.436	0.062	8.2	73.8
A6 Debate	Architecture	0.471	0.079	5.1	75.8

D Detailed Controlled Intervention Results

The main paper reports the headline intervention effects. This section gives the selection rationale, realized-execution audits, complete role-swap coverage, and permission enforcement and evaluator-validity checks needed to bound those claims.

D.1 Topology Intervention: Configured and Realized Semantics

The experiment configures Chain, Star, and DAG edge sets over the same five positions. A post-hoc runtime audit confirms exact configured routing for Chain and DAG. In the configured-Star condition, empty-upstream positions inherit accumulated global history, so this condition does not realize independent peripheral processing or a bidirectional Finalizer-centered star. We therefore

use Chain–DAG as the validated routing contrast and retain the configured-Star condition only to document executor sensitivity.

Table 15: Configured topology conditions and their realized interpretation.

Condition	Configured motif	Realized interpretation
Chain	Serial path	Exact configured predecessor sets
DAG	Branch–merge graph	Exact configured predecessor sets
Configured Star	Finalizer-centered graph	Sequential accumulation from global-history fallback

Table 16: Configuration and coverage of the topology intervention.

Factor	Setting
Role set	Planner, Coder, Verifier, Critic, Finalizer
Topology conditions	Chain, Star, DAG
Benchmarks	AIME 2026, GPQA Diamond, HLE, ARC-AGI-2, IFBench
Attribution	LOO with null-agent replacement
Complete-system runs	$5 \times 30 \times 3 \times 3 = 1{,}350$
LOO records	6,750

Table 17: Realized execution audit for every topology condition.

Trace property	Observed
Role messages per run	5
Evaluator events per run	1
Execution passes	1
Dynamic feedback rounds	0
Parallel DAG execution	No
Hub dispatch or mediation	No
Topology-specific trace edges	No
Finalizer executed last	Yes

Chain and DAG match their configured predecessor sets at all 4,500 audited node inputs. The configured-Star condition mismatches 1,800/2,250 inputs: empty-upstream positions inherit accumulated prior messages, while Finalizer receives all four peripheral outputs. The runner reconstruction was checked against the 1,353 recoverable raw prompts (20.0% of inputs), with zero disagreement. Remaining inputs were deterministically reconstructed from the archived graph, execution order, traces, and message-construction code. Accordingly, causal routing claims use only Chain–DAG; configured Star is an implementation diagnostic.

Paired uncertainty analysis. We estimate routing-induced LOO changes using 20,000 stratified task-cluster bootstrap replicates with seed 20260923. Tasks are resampled within each benchmark, retaining their three execution seeds and all paired routing conditions. Percentile intervals contain the central 95% of bootstrap differences.

Table 18: Paired changes in position-level LOO under routing conditions. Intervals use 20,000 task-cluster bootstrap replicates. Chain–DAG is the runtime-validated routing contrast. Configured-Star contrasts are descriptive executor diagnostics because that condition exposes accumulated global history.

Role	Contrast	ΔLOO	95% CI
Runtime-validated routing contrast			
Verifier	**DAG − Chain**	$\mathbf{-0.091}$	$\mathbf{[-0.147, -0.040]}$
Critic	DAG − Chain	-0.022	$[-0.053, +0.009]$
Finalizer	DAG − Chain	-0.027	$[-0.056, +0.002]$
Configured-Star executor diagnostics			
Verifier	Star − Chain	-0.042	$[-0.087, 0.000]$
Verifier	Star − DAG	$+0.049$	$[+0.011, +0.089]$
Critic	Star − Chain	-0.053	$[-0.098, -0.011]$
Critic	Star − DAG	-0.031	$[-0.073, +0.011]$
Finalizer	Star − Chain	$+0.009$	$[-0.022, +0.040]$
Finalizer	Star − DAG	$+0.036$	$[+0.002, +0.071]$

Verifier mean LOO lies on opposite sides of zero in the two validated conditions: $+0.051$ (95% CI $[+0.016, +0.089]$) in Chain and -0.040 ($[-0.076, -0.007]$) in DAG. Critic decreases from $+0.051$ ($[+0.013, +0.089]$) to $+0.029$ ($[-0.007, +0.067]$), but its paired interval includes zero.

Finalizer remains strongly positive in both conditions: $+0.380$ ($[+0.311, +0.449]$) and $+0.353$ ($[+0.284, +0.422]$), respectively.

Aggregate versus task-level sign changes. The aggregate sign reversal does not imply that most tasks individually flip. After averaging the three seeds within each task, Verifier LOO has opposite nonzero signs under Chain and DAG for 12/150 tasks (8.0%). Because LOO is sparse, only 19 tasks are nonzero in both conditions; among these, 12/19 (63.2%) exhibit a strict sign flip. The aggregate reversal therefore reflects a redistribution of positive, negative, and zero marginal events across tasks, rather than a universal per-task transition.

D.2 Role-Prompt Intervention: Complete Swap Coverage

The intervention exchanges functional prompts while holding graph positions, position permissions, topology, model, tasks, and seeds fixed. Planner–Coder is evaluated in A2, A3, and A5; Coder–Verifier and Critic–Finalizer are evaluated in A5.

The selected swaps are diagnostic rather than exhaustive. They cover three functionally distinct boundaries in the workflow: planning versus implementation, generation versus verification, and critique versus final synthesis. Each contrast tests whether measured contribution follows the functional prompt or remains attached to its graph position and downstream dependencies.

Table 19: Rationale for the selected role-prompt swaps.

Swap	Functional contrast	Diagnostic purpose
Planner ↔ Coder	Planning vs. implementation	Tests whether assigning decomposition and artifact construction to their usual positions is consequential across different information routes.
Coder ↔ Verifier	Generation vs. verification	Tests whether contribution follows solution generation or tool-supported checking when the prompts are exchanged at fixed positions.
Critic ↔ Finalizer	Critique vs. final synthesis	Separates the functional effect of critique and synthesis prompts from the dependence created by the terminal output position.

Together, these contrasts cover upstream, intermediate, and terminal handoffs, where an intrinsic-role interpretation makes distinct predictions.

After filtering the A5 archive to its final execution fingerprint, $12{,}600$ of $15{,}368$ raw attribution records remain, exactly matching four conditions, 450 runs per condition, and seven positions per run. Earlier partial records are excluded from every result below.

Table 20: Role-prompt coverage audit. A5 contains one matched baseline and all three completed swap conditions.

Architecture and coverage block	Positions	Conditions	Full runs	LOO records
A2 Planner–Coder	4	2	900	3,600
A3 Planner–Coder	6	2	900	5,400
A5 baseline plus three swaps	7	4	1,800	12,600

Table 21: Paired complete-team outcomes for every architecture–swap condition. Confidence intervals bootstrap task clusters and retain all seed observations within each sampled task.

Swap	Architecture	Pairs	Improved	Worsened	Unchanged	ΔU	95% CI
Planner–Coder	A2 Chain	450	33	32	385	$+0.002$	$[-0.036, +0.040]$
Planner–Coder	A3 DAG	450	30	19	401	$+0.024$	$[-0.009, +0.058]$
Planner–Coder	A5 Star	450	30	31	389	-0.002	$[-0.038, +0.033]$
Coder–Verifier	A5 Star	450	31	28	391	$+0.007$	$[-0.024, +0.038]$
Critic–Finalizer	A5 Star	450	46	29	375	$+0.038$	$[-0.007, +0.087]$

Table 22 reports the LOO change at every graph position, not only the two positions whose prompts were exchanged. This full matrix is necessary because a local prompt edit can change the downstream state and move marginal value to an untouched position.

Table 22: Complete position-level LOO changes, $\Delta\mathrm{LOO}$ = swapped minus matched baseline. A dash means that the position is absent from the architecture.

Position	P–C, A2	P–C, A3	P–C, A5	C–V, A5	Crit–F, A5
Planner	$+0.040$	$+0.029$	$+0.036$	-0.009	$+0.020$
Coder	$+0.002$	$+0.024$	-0.036	$+0.011$	$+0.011$
Reviewer	$+0.002$	–	–	–	–
Retriever	–	$+0.029$	–	–	–
Debugger	–	$+0.024$	–	–	–
Tester	–	$+0.024$	–	–	–
Verifier	–	–	$+0.031$	$+0.031$	$+0.000$
Critic	–	–	$+0.002$	$+0.000$	$+0.004$
Researcher	–	–	-0.007	-0.013	$+0.016$
Supervisor	–	–	-0.002	$+0.004$	$+0.176$
Finalizer	$+0.002$	$+0.020$	$+0.000$	$+0.011$	$+0.040$

Table 23: Benchmark heterogeneity for every role-prompt swap. Planner–Coder uses 270 pairs per benchmark across A2/A3/A5; the A5-only swaps use 90 pairs. Cells report ΔU and task-cluster bootstrap 95% intervals.

Benchmark	Planner–Coder	Coder–Verifier	Critic–Finalizer
AIME 2026	$+0.033$ $[-0.048, +0.115]$	$+0.000$ $[-0.111, +0.111]$	$+0.200$ $[+0.078, +0.333]$
GPQA Diamond	-0.019 $[-0.081, +0.033]$	$+0.022$ $[-0.056, +0.100]$	$+0.089$ $[-0.022, +0.200]$
HLE	$+0.000$ $[-0.015, +0.019]$	$+0.022$ $[-0.022, +0.078]$	$+0.056$ $[-0.011, +0.133]$
ARC-AGI-2	$+0.007$ $[-0.033, +0.059]$	-0.022 $[-0.067, +0.022]$	-0.011 $[-0.033, +0.000]$
IFBench	$+0.019$ $[-0.026, +0.059]$	$+0.011$ $[-0.022, +0.056]$	-0.144 $[-0.256, -0.033]$

The complete coverage rejects a simple "contribution follows the prompt" account. Planner–Coder in A5 produces an almost antisymmetric redistribution between the two exchanged positions without improving team utility. More strikingly, Critic–Finalizer increases untouched Supervisor LOO by 0.176, far more than the changes at Critic ($+0.004$) or Finalizer ($+0.040$). Its mean utility effect also combines an AIME gain ($+0.200$) with an IFBench loss (-0.144). Prompt interventions therefore reorganize a global execution state rather than carrying a fixed amount of contribution with a local role label.

D.3 Runtime Permissions: Complete Enforcement and Evaluator Audit

The main A2/A3 experiment toggles one permission at a time in both directions; the A5 experiment separately revokes or reassigns authority. Permissions are runtime enforced rather than prompt-only instructions. Tables below separate four questions that should not be conflated: whether the override was applied, whether an attempted action was blocked, which answer source the router selected, and whether the external evaluator was valid.

Table 24: Complete A2/A3 permission enforcement controls. Each row aggregates both architectures and five benchmarks.

Condition	Runs	Override check	Blocked-action audit	Realized answer source
`write_solution=false`	900	900/900	Coder solution artifact blocked in 900/900	Finalizer, 900/900
`write_solution=true`	900	900/900	No Coder artifact block in 900/900	Finalizer, 900/900
`final_answer=false`	900	900/900	No-authorized-answer flag in 900/900	No selected source, 900/900
`final_answer=true`	900	900/900	No no-authorized-answer flag in 900/900	Finalizer, 900/900

These enforcement counts retain ARC-AGI-2 because they audit whether the runtime permission was applied, not whether the resulting evaluator path satisfies the utility definition in equation 2. The primary A2/A3 `final_answer` utility analysis therefore uses only the 720 non-ARC matched pairs, while all 900 runs remain part of the enforcement audit.

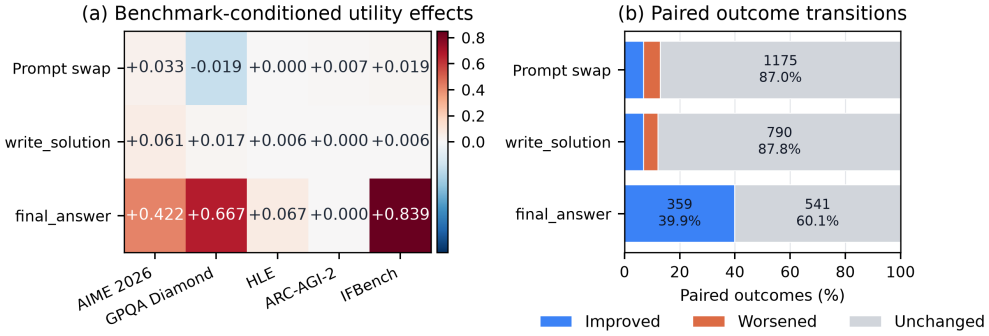


Figure 16: Benchmark-conditioned utility effects and paired transitions for the main prompt and permission interventions. Positive values favor the swapped or enabled condition. ARC-AGI-2 under `final_answer=false` is shown only as an evaluator-path diagnostic and is excluded from the primary A2/A3 `final_answer` utility contrast because its evaluator can score a retained artifact despite the absence of an authorized final answer.

Enabling `write_solution` improves 63 of 900 pairs, worsens 47, and leaves 790 unchanged. For `final_answer`, the primary utility analysis excludes ARC-AGI-2 and contains 720 matched non-ARC pairs. Under the disabled condition, all 720 runs have no selected evaluator-visible source and therefore receive zero utility under equation 2. Enabling `final_answer` raises mean utility from 0.000 to 0.499, improving 359 pairs (49.9%), worsening none, and leaving 361 unchanged (50.1%). The enforcement logs establish that the first contrast blocks only the solution artifact, whereas the second restores the sole selected evaluator-visible output path.

Table 25: A5 permission conditions relative to the matched baseline. Utility effects use the four benchmarks with valid evaluators and therefore contain 360 pairs per contrast.

Condition	Pairs	Improved	Worsened	Unchanged	ΔU	95% CI
Revoke Coder write	360	16	28	316	-0.033	$[-0.069, +0.003]$
Assign write to Verifier	360	23	27	310	-0.011	$[-0.050, +0.031]$
Revoke Finalizer answer	360	18	9	333	$+0.025$	$[-0.006, +0.056]$
Reassign Finalizer answer to Verifier	360	23	23	314	$+0.000$	$[-0.039, +0.039]$

Table 26: Complete A5 full-run enforcement and output-routing audit. Each condition contains 450 complete runs before evaluator exclusions.

Condition	Applied override	Blocked output	Fallback	Selected source
Baseline	None	None	0/450	Finalizer, 450/450
Revoke Coder write	Coder write: false, 450/450	Coder artifact, 450/450	0/450	Finalizer, 450/450
Assign write to Verifier	Coder false; Verifier true, 450/450	Coder artifact, 450/450	0/450	Finalizer, 450/450
Revoke Finalizer answer	Finalizer answer: false, 450/450	None	450/450	Supervisor, 450/450
Reassign answer to Verifier	Finalizer false; Verifier true, 450/450	None	450/450	Supervisor, 450/450; Verifier, 0/450

Table 27: Evaluator-validity and archive-completeness audit. These checks are reported separately from permission enforcement because a correctly enforced condition can still have an invalid evaluator path.

Scope	Audit result	Analysis treatment
A2/A3, non-ARC, answer disabled	Across 720 full runs, no source is selected and all 720 evaluator-visible outputs are empty	Included in the primary A2/A3 `final_answer` contrast; disabled utility is zero under equation 2
A2/A3, ARC-AGI-2, answer disabled	All 180 runs have the `no-authorized-answer` flag, yet retained artifacts remain scorable: 32 score one and 148 score zero	Excluded from the primary A2/A3 `final_answer` utility analysis because this evaluator path bypasses the output-validity convention; retained only as an evaluator-path diagnostic
A5 archive completeness	The 15,750 attribution and 18,000 coalition records exactly match $5 \times 450 \times 7$ and $5 \times 450 \times 8$, respectively	Retained after the unique-ID audit
A5, four valid benchmarks	All 14,400 coalition evaluations have valid outcomes from successfully loaded official evaluators	360 paired full runs per contrast retained
A5 IFBench	Of 3,600 coalition evaluations, 3,060 have evaluator import errors and 540 have empty outputs	Entire benchmark excluded from A5 utility contrasts

The audit supports two distinct conclusions. First, write authority is not work access: revoking Coder's artifact permission is enforced in every run, yet ordinary messages remain available and most task outcomes do not change. Second, nominal answer authority is not realized control. A5 already authorizes Supervisor, so revoking Finalizer activates a deterministic fallback; granting Verifier authority does not make Verifier the selected source in any of 450 runs. These conclusions concern artifact publication and answer submission only. They do not identify the effects of general tool, sandbox, or message access, and permission-conditioned LOO remains a mechanism diagnostic rather than a standalone capability measure.