MAS-GATE: UNLOCKING MULTI-AGENT SYSTEM LIFECYCLE VIA NODE-CENTRIC QUANTIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

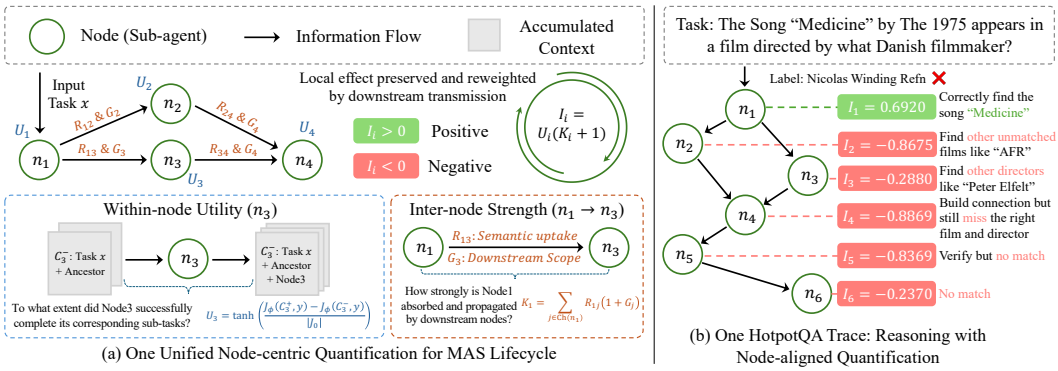
Large language model (LLM)-based multi-agent systems (MAS) can solve complex problems beyond the reach of a single agent by decomposing tasks and coordinating complementary capabilities. Despite extensive research on MAS, existing work largely focuses on specific aspects or individual phases of their development and operation, resulting in fragmented perspectives with objectives and signals that are difficult to reconcile across these different settings. To bridge these disconnected perspectives, we conceptualize MAS development and operation as a three-phase lifecycle spanning design-time orchestration *before execution*, runtime intervention *during execution*, and post-hoc attribution *after execution*, and introduce MAS-Gate, a unified node-centric quantification applicable across these phases. Modeling each execution as a directed acyclic graph, MAS-Gate combines within-node utility, which estimates a node's contribution through changes in reference-answer likelihood, with inter-node strength, which approximates semantic uptake and potential downstream reach. We adapt this shared quantification to three objectives: providing a structure-aware dense reward for Orchestrator training, training an Editor to act on execution prefixes using paired rewards from completed edited and unedited trajectories, and supplying risk-ranked nodes as auxiliary evidence for LLM-based post-hoc attribution. Experiments spanning interaction environments, knowledge-intensive multi-hop question answering, general reasoning, and post-hoc attribution demonstrate that MAS-Gate improves workflow construction, targeted runtime editing, and LLM-assisted error localization over corresponding phase-specific baselines. Together, these results establish MAS-Gate as a unified structural interface for improving MAS generation, dynamic editing, and error attribution throughout the lifecycle.

1 INTRODUCTION

Multi-agent systems (MAS) have emerged as an effective way to improve the performance of large language models (LLMs) on complex tasks (Ke et al., 2025a; Huang et al., 2026; Tran et al., 2025; Lin et al., 2026a; Jwalapuram et al., 2026). By decomposing a problem, assigning specialized roles, and coordinating intermediate results, an MAS can elicit complementary capabilities that are difficult to obtain from a single model invocation (Yang et al., 2026; Lin et al., 2026b; Chen et al., 2024; Cemri et al., 2026). This potential has motivated research on many aspects of MAS, including workflow construction, topology search, model routing, communication pruning, and failure diagnosis in multi-agent executions (Zhang et al., 2025c; Zhuge et al., 2024; Yue et al., 2025; Liu et al., 2023). Despite this breadth, existing methods remain largely siloed, with most optimizing or analyzing a particular phase through a phase-specific objective, representation, or heuristic (Hu et al., 2025; Ke et al., 2025b). This specialization produces signals tailored to individual objectives, with limited correspondence across orchestration, intervention, and attribution (Qi et al., 2026).

To organize these phase-specific objectives, we define the MAS lifecycle as an integrated framework spanning the initial construction of an MAS, its dynamic adjustment during execution, and the attribution of failures after execution (Li et al., 2024; Qi et al., 2026). Specifically, *Before execution (design-time orchestration)*, the goal is to construct an effective MAS by organizing its agents and their interactions (Fan et al., 2025; Ruan et al., 2026). *During execution (runtime intervention)*, the goal is to monitor the partial trajectory and selectively adjust the active MAS, e.g.,

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.



(a) One Unified Node-centric Quantification for MAS Lifecycle

(b) One HotpotQA Trace: Reasoning with Node-aligned Quantification

Figure 1: Overview of MAS-Gate. (a) Unified node-centric quantification combines within-node utility and inter-node strength to measure node impact across the MAS lifecycle. (b) A HotpotQA trace illustrates how node impact scores distinguish useful contributions from erroneous reasoning.

adding or removing sub-agents, replacing an agent, or rerouting dependencies without regenerating the entire workflow. *After execution (post-hoc attribution)*, the goal is to analyze the completed wrong execution and identify the components responsible for its failure (Xia et al., 2026; Zhu et al., 2025). Existing research has concentrated primarily on the first and third phases, whereas runtime intervention remains comparatively underexplored. Some systems respond to execution failures by regenerating or refining the entire MAS (Wang et al., 2026a), but we focus on using the accumulated trajectory to selectively and dynamically adjust the continuation of an active multi-agent execution.

In this paper, we introduce MAS-Gate, a unified node-centric quantification that provides a common formulation for the distinct objectives of the MAS lifecycle. Following previous works (Zhuge et al., 2024), we model the MAS as a directed acyclic graph to expose dependencies among agent computations. Each node represents a sub-agent, and each directed edge records which upstream outputs are available to a later sub-agent. MAS-Gate then quantifies two complementary properties of this execution through a *Node-Centric Quantification*. First, within-node utility U_i quantifies the change in reference-answer likelihood induced by incorporating node i's output, serving as a proxy for its contribution to task progress. Second, inter-node strength K_i combines estimated semantic uptake with potential downstream reach. The resulting node impact $I_i = U_i(1+K_i)$ integrates these complementary perspectives while preserving local utility when $K_i = 0$. Figure 1(a) summarizes this quantification, while Figure 1(b) illustrates how the resulting node impact scores reflect the quality of a HotpotQA trace. In this example, the initial step correctly identifies the song and receives a positive impact score, whereas subsequent steps pursue incorrect film and director associations and fail to recover the correct answer, receiving negative scores. MAS-Gate thus distinguishes a useful intermediate contribution from erroneous reasoning within the same failed execution.

We evaluate the same MAS-Gate quantification across all three phases of the MAS lifecycle. For design-time orchestration, we combine mean node impact with terminal correctness as a dense reward to train an Orchestrator, improving the initial MAS workflow construction ability. For runtime intervention, we train an Editor to condition on the current MAS graph and accumulated execution trace, applying targeted adjustments to the remaining workflow after each node completes and before its output propagates downstream. For post-hoc error attribution, we derive risk scores from node utility and inter-node strength to rank the top-k suspicious steps or agents as auxiliary evidence for an LLM localizer. The localizer retains access to the complete execution trajectory and is free to identify candidates beyond the supplied ranking.

Across experiments, Qwen3-8B (Team, 2025) serves as the backbone for training the Orchestrator and Editor, Qwen3-30B-A3B-Instruct-2507 serves as the frozen sub-agent model for sub-task execution and within-node utility U_i quantification, and bge-m3-zeroshot-v2.0 (Laurer et al., 2023) provides the entailment scores used to quantify inter-node strength K_i. The evaluation covers interaction environment, multi-hop question answering, general reasoning, and post-hoc error attribution, with the first three task families supporting the design-time and runtime studies and the last supporting the attribution experiment. MAS-Gate outperforms existing orchestration baselines in

task performance and generalization, and improves failure localization over direct LLM-based attribution. Beyond these established settings, our runtime intervention study demonstrates that targeted adjustments to an active MAS can improve execution outcomes without reconstructing the entire workflow, supporting dynamic intervention as an effective means of refining multi-agent execution. Together, these findings support the use of a common quantification mechanism to guide distinct objectives across the MAS lifecycle. Our contributions are as follows:

- We formulate the MAS lifecycle as a unified framework encompassing design-time orchestration, runtime intervention, and post-hoc attribution, connecting different existing MAS research.
- We introduce MAS-Gate, a unified node-centric quantification method that combines within-node utility and inter-node strength, providing a shared quantitative signal that can be adapted across all three lifecycle phases.
- We evaluate MAS-Gate across interactive environments, multi-hop question answering, general reasoning, and failure attribution, demonstrating gains over existing baselines in orchestration and failure localization, as well as the effectiveness of targeted intervention during execution.

2 Related Work

2.1 The Multi-Agent System Lifecycle

Before Execution: Design-Time Orchestration. Design-time orchestration constructs the agents, roles, communication structure, and execution order that will carry out a task (Guo et al., 2024; Wu & Shu, 2026; Qi et al., 2026). Existing approaches can be broadly divided into inference-time orchestration and training-time orchestration (Ke et al., 2025a). Inference-time methods generate or search for a workflow without updating the underlying agent models. They use language-model-guided workflow synthesis, heuristic or evolutionary search, role and model assignment, and graph or topology construction to determine the decomposition and dependencies of an MAS (Zhang et al., 2025c; Ruan et al., 2026; Ke et al., 2025b; Li et al., 2026; Zhang et al., 2024). Training-time orchestration instead trains a dedicated orchestrator or workflow policy to emit, score, or refine MAS specifications from task feedback and execution traces. Its representative routes include orchestration-trace policy optimization, orchestration-level reward modeling, counterfactual credit assignment for workflow decisions, and trainable architecture or topology search (Zhang, 2026; Chen et al., 2026b; Tsang et al., 2026; Ke et al., 2026; Zhang et al., 2025a; Shi et al., 2026). These approaches seek to learn or construct an orchestrator that produces more effective multi-agent workflows.

During Execution: Runtime Intervention. Runtime intervention adjusts an ongoing execution using intermediate observations. Self-Refine improves an existing output through iterative feedback and revision by the same LLM (Madaan et al., 2023). AgentDropoutV2 intercepts agent outputs and uses retrieved failure indicators to rectify erroneous messages or reject those that remain unreliable (Wang et al., 2026b). Related approaches optimize agent communication: DyLAN combines agent-team selection with dynamic collaboration (Liu et al., 2023), while AgentPrune performs one-shot pruning of spatial and temporal communication dependencies (Zhang et al., 2025b). Building on these directions, We study runtime intervention through an Editor that conditions on the current MAS graph and accumulated execution trace to selectively revise the remaining workflow while preserving unaffected computation. The central challenge is to determine whether and where intervention is needed as weaknesses emerge during execution, enabling targeted adjustments before problematic outputs propagate downstream.

After Execution: Post-Hoc Attribution. Post-hoc attribution focuses on failed MAS trajectories and seeks to identify the agent, node, or dependency responsible for the observed error. The Who&When benchmark is designed specifically for this setting, pairing multi-agent execution traces with the question of which agent introduced or propagated the failure (Zhang et al., 2025d). AEGIS complements this setting by injecting controlled errors into successful multi-agent executions and annotating the faulty agents and their error modes (Kong et al., 2026). Related methods use spectrum-based analysis, trace-level failure localization, failure-mode taxonomies, graph-based execution analysis, and counterfactual contribution estimates to rank suspicious agents or interactions (Xia et al., 2026; Kong et al., 2026; Ge et al., 2026; Ma et al., 2026). The central output of this phase is an attribution of responsibility within an already completed and erroneous execution.

Lifecycle Perspective. Existing methods are typically developed and evaluated within individual lifecycle phases, yielding representations and signals tailored to their respective objectives. These outputs are therefore not directly interchangeable across orchestration, intervention, and attribution. MAS-Gate connects these settings through a common node-scoring formulation.

2.2 Quantification of Multi-Agent Systems

Prior work quantifies multi-agent interaction with information-theoretic measures of uncertainty and information exchange, as well as graph-based measures of diversity, redundancy, communication efficiency, and structural connectivity (Kim et al., 2025; Yang et al., 2026; Lee et al., 2025; Shen et al., 2025; Chen et al., 2026a; Rath, 2026). Quantification methods based on repeated rollouts or counterfactual interventions require additional MAS executions to estimate agent contributions, introducing computational overhead that can become prohibitive during large-scale training or latency-sensitive deployment. In contrast, MAS-Gate provides node-centric quantification through a single forward pass over the observed execution trace, without re-executing the MAS.

3 Node-Centric Quantification

3.1 Execution Representation

Given a task x and reference answer y, we represent an MAS execution as a directed acyclic graph $\mathcal{G} = (V, E)$. Each node $i \in V$ corresponds to an accountable sub-agent that produces output o_i, and each edge $(i, j) \in E$ indicates that o_i is available to node j. The resulting execution trace is

$$\tau = \big(x, y, \mathcal{G}, \{o_i\}_{i \in V}, o_{\text{final}}\big), \tag{1}$$

where o_{final} denotes the system's final response. This representation captures both node-level outputs and the dependencies through which they may influence downstream computation. Building on this representation, MAS-Gate characterizes each execution along two complementary dimensions. Within-node utility estimates local contribution through changes in reference-answer support, whereas inter-node strength captures semantic uptake and potential downstream reach. Their combination yields a node-level impact score that relates local utility to the semantic and structural context of subsequent computation.

3.2 Within-Node Utility Quantification

Dynamically generated subtasks rarely admit gold-standard intermediate labels, making it difficult to assign reliable dense rewards to individual sub-agents. Terminal correctness provides only sparse execution-level supervision and does not reveal whether a particular node moves the accumulated trajectory toward or away from the reference answer. We therefore adopt a perplexity-guided reward as a proxy to measure this effect (Lin et al., 2026b; Liu et al., 2026). The underlying premise is that a useful node output should make the reference answer y easier to predict from the evidence accumulated so far, whereas a harmful output should have the opposite effect.

Let C denote the accumulated evidence available to a sub-agent, and let p_ϕ denote the sub-agent LLM. We use this LLM to evaluate the predictability of the reference answer y given C and define

$$J_\phi(C, y) = \frac{1}{|y|} \sum_{t=1}^{|y|} \log p_\phi(y_t \mid C, y_{<t}), \tag{2}$$

$$\mathrm{PPL}_\phi(y \mid C) = \exp(-J_\phi(C, y)), \tag{3}$$

where $J_\phi(C, y)$ is the mean token-level log-likelihood assigned by the sub-agent LLM to y conditioned on C. A larger $J_\phi(C, y)$ corresponds to lower perplexity and indicates that the accumulated evidence better supports the reference answer. Let $\mathrm{Anc}(i)$ denote the set of ancestors of node i in $\mathcal{G}$. For node i, the pre-node context contains the task and the outputs of these ancestors, while the post-node context additionally includes o_i:

$$C_i^- = \text{Concat}(x, \{o_a : a \in \mathrm{Anc}(i)\}), \tag{4}$$

$$C_i^+ = \text{Concat}(x, \{o_a : a \in \mathrm{Anc}(i)\}, o_i). \tag{5}$$

To normalize the change against a task-only reference, we define $J_0 = J_\phi(x, y)$ to measure the sub-agent LLM's baseline log-likelihood for the reference answer before any agent output is provided. The within-node utility is then defined as

$$U_i = \tanh\left(\frac{J_\phi(C_i^+, y) - J_\phi(C_i^-, y)}{|J_0|}\right). \tag{6}$$

If adding o_i increases the log-likelihood of y, it lowers the corresponding perplexity and produces a larger U_i, and vice versa. Normalization by the task-only baseline improves comparability across tasks, while the $tanh$ function smooths outliers and preserves the ordering of node contributions.

3.3 Inter-Node Strength Quantification

Within-node utility characterizes local reference-answer support but does not account for the node's downstream context. Inter-node strength incorporates this context through two factors: estimated semantic uptake by each child j, denoted by R_{ij}, and the downstream scope, denoted by G_j.

For each edge $(i, j) \in E$, we use entailment between the upstream output o_i and downstream output o_j as a semantic proxy for uptake:

$$R_{ij} = p_{\mathrm{NLI}}(\text{entailment} \mid o_i, o_j), \qquad R_{ij} \in [0, 1]. \tag{7}$$

where p_{NLI} is an entailment model and R_{ij} is the entailment probability for the pair (o_i, o_j). We also consider its potential downstream reach. Let $\mathrm{Ch}(i)$ and $\mathrm{Desc}(i)$ denote the sets of children and descendants of node i, respectively. We define the propagation scope of node j as

$$G_j = \frac{|\,\mathrm{Desc}(j)|}{\max_{v \in V} |\,\mathrm{Desc}(v)|}. \tag{8}$$

Since each execution graph contains at least one dependency edge, the denominator is strictly positive. $G_j = 0$ for a terminal node indicates no further reach, rather than no semantic uptake at that node. We therefore apply a unit baseline shift to G_j, so that an edge ending at a terminal child still contributes R_{ij} to the strength of its predecessor. The resulting inter-node strength is

$$K_i = \sum_{j \in \mathrm{Ch}(i)} R_{ij}(1 + G_j). \tag{9}$$

We then combine local utility and inter-node strength to define node impact. A second unit baseline shift is introduced at this stage to preserve the node's local contribution independently of downstream transmission: $I_i = U_i(1 + K_i)$. A node with large U_i and K_i provides useful information that is broadly incorporated into subsequent computation. In contrast, a node with low or negative U_i but large K_i indicates weak or harmful information that is amplified downstream.

4 Lifecycle Applications of Node-Centric Quantification

4.1 Design-Time Orchestration

4.1.1 Experimental Settings

Dense reward for Orchestrator training. For an execution trace τ with realized node set V, we average the node impacts to obtain an MAS-level dense reward and combine it with binary answer correctness:

$$r(\tau) = \frac{1}{|V|} \sum_{i \in V} I_i + c(\tau), \tag{10}$$

where $c(\tau) \in \{0, 1\}$ denotes the terminal accuracy reward. The resulting objective retains outcome-level supervision while providing a graded signal that reflects the contributions of individual nodes throughout the execution. We use this fused reward with standard group-relative policy optimization (GRPO) (Shao et al., 2024) to train the Orchestrator, while leaving the sub-agents and quantification models fixed. Appendix D.1.2 compares this fused reward with an accuracy-only reward, demonstrating the contribution of MAS-Gate quantification to orchestration performance and out-of-task generalization.

Table 1: Quantification comparison of MAS-Gate against baselines on design-time orchestration. ALFW, SciW, VITA, BCP, and GPQA denote ALFWorld, SciWorld, VitaBench, BrowseComp-Plus, and GPQA-Diamond, respectively. **Bold** indicates the best result.

Category	Method	Train: ALFW			Train: HotpotQA		Train: DeepScaler		Avg
		ALFW	SciW_{OOT}	VITA_{OOT}	HotpotQA	BCP_{OOT}	AIME	GPQA_{OOT}	
General	CoT	37.18	1.26	29.19	50.00	11.31	28.07	48.99	29.43
	CoT-SC	32.05	1.07	28.23	54.00	8.33	29.82	52.02	29.36
Inference-time	AFlow	31.09	3.90	23.54	46.50	12.50	15.79	33.84	23.88
	AOrchestra	57.05	8.93	9.18	48.00	12.50	40.35	57.07	33.30
	Skill-MAS	28.21	5.25	13.55	54.00	8.93	19.30	48.99	25.46
Training-time	MAS-GPT	53.53	14.81	7.52	51.00	11.31	24.56	54.04	30.97
	ToolOrchestra	37.50	7.69	10.98	37.50	3.57	12.28	36.36	20.84
	MAS-Orchestra	14.42	2.33	22.22	51.00	9.52	21.05	41.41	23.14
MAS-Gate	Orchestrator	**57.69**	**18.27**	**45.29**	**69.50**	**13.69**	**47.37**	**61.11**	**44.70**

Benchmarks. We consider three task families: interaction environment, multi-hop question answering, and general reasoning. The Orchestrator is trained on ALFWorld (Shridhar et al., 2020), HotpotQA (Yang et al., 2018), and DeepScaler (Tan et al., 2025). We evaluate its in-task performance on ALFWorld, HotpotQA, and AIME (MAA-Committees, 2025), and its out-of-task (OOT) generalization on SciWorld (Wang et al., 2022a), BrowseComp-Plus (Chen et al., 2025), and GPQA-Diamond (Rein et al., 2023). Here, OOT evaluation assesses transfer to benchmarks not used for Orchestrator training. All metrics are reported as 0-100% percentages. Appendix C.1 provides more details on the datasets, data splits, and evaluation protocols.

Baselines. We compare MAS-Gate against three baseline families. (1) CoT (Wei et al., 2022) and CoT-SC (Wang et al., 2022b) serve as general inference baselines; (2) AFlow (Zhang et al., 2025c), AOrchestra (Ruan et al., 2026), and Skill-MAS (Lin et al., 2026c) represent inference-time workflow construction; and (3) MAS-GPT (Ye et al., 2025), ToolOrchestra (Su et al., 2025), and MAS-Orchestra (Ke et al., 2026) represent training-time approaches that learn multi-agent coordination. Detailed descriptions and adaptation settings for all baselines are illustrated in Appendix C.2.

4.1.2 Results and Analysis

First, MAS-Gate delivers consistent performance gains over both single-agent and multi-agent baselines. As shown in Table 1, existing MAS baselines do not consistently outperform single-agent baselines. For example, AOrchestra surpasses CoT on ALFWorld and ScienceWorld but falls below it on VitaBench. Such variation is consistent with prior findings that architectural inefficiencies can prevent automatically constructed MAS from realizing the benefits of multi-agent coordination (Jwalapuram et al., 2026). In contrast, MAS-Gate achieves the best results across all seven evaluation benchmarks, reaching an average score of 44.70 against 33.30 for the strongest competing MAS method. These gains support the value of MAS-Gate quantification that guides the Orchestrator toward more effective multi-agent organization.

Second, MAS-Gate's orchestration policy generalizes beyond its training benchmarks. It achieves the best performance on all four OOT benchmarks, with the largest margin over competing MAS methods on VitaBench. On this benchmark, all six MAS baselines fall below CoT, whereas MAS-Gate reaches 45.29%, surpassing the strongest MAS baseline by 21.75%. This contrast highlights the limited transferability of existing MAS methods, whose advantages do not reliably carry over to unseen benchmarks. With MAS-Gate quantification as a structure-aware training reward, the Orchestrator instead maintains a consistent advantage across OOT settings. Taken together, these results support the transferability of MAS-Gate-trained orchestration across the evaluated task settings, without identifying the specific coordination mechanisms underlying this advantage.

4.2 RUNTIME INTERVENTION

4.2.1 EXPERIMENTAL SETTINGS

Paired reward for Editor training. We use MAS-Gate to train an Editor for selective runtime intervention, keeping the Orchestrator and sub-agents fixed. After each node completes and before downstream nodes consume its output, the Editor observes the task, the current MAS graph, and the accumulated execution outputs. It can choose from six actions, grouped by their position relative to the current node: (1) *Current-node decisions*: leave execution unchanged (KEEP), retry the current node, or append a bounded diagnostic annotation to the current node's output; (2) *Upstream intervention*: re-execute a parent and its affected descendants; and (3) *Downstream intervention*: guide an unexecuted child or insert an auxiliary sub-task into the remaining workflow. All actions are restricted to legal targets and preserve unaffected computation. To balance computational efficiency and execution stability, we allow at most one non-KEEP intervention over the entire execution trace.

During training, we compare an edited execution τ^a with a cached no-intervention reference τ^0 for the same task and initial workflow. Both executions share the observed prefix up to the intervention decision. The Editor selects actions from the observed prefix, whereas the paired reward is evaluated after both continuations are complete. The decision interface is specified in terms of the task, graph, and observed outputs; reference answers and quantification scores are reserved for reward computation (Appendix C.3.2). We aggregate the shared node impacts as $\bar{I}(\tau) = |V_\tau|^{-1} \sum_{i \in V_\tau} I_i$ where V_τ is the realized node set. A valid non-KEEP action a receives the paired reward

$$r(a) = c(\tau^a) - c(\tau^0) + \beta\big[\bar{I}(\tau^a) - \bar{I}(\tau^0)\big] - \lambda C(a). \tag{11}$$

where $c(\tau)$ is the task outcome score, $C(a)$ is a predefined action-dependent intervention penalty, and β and λ control the quantification and cost terms. This reward balances task improvement, changes in node-level contributions, and intervention cost. Each KEEP decision receives zero local reward, while invalid actions receive a fixed negative penalty. We train the Editor using GRPO (Shao et al., 2024) with multiple decision sequences sampled for each task.

Benchmarks. We adopt the training and evaluation benchmarks from Section 4.1, excluding VitaBench. It involves multi-turn interactions, and each turn requires multiple tool calls, which does not readily support the paired comparisons required here, where the no-intervention and Editor branches must share the same initial workflow and pre-intervention execution prefix. Further details on the benchmarks and data splits are provided in Appendix C.1.

Baselines. We compare against three baseline families. (1) *LLM Editors* use DeepSeek-V4-Flash and GPT-5.4-Mini as frozen prompted Editors to intervene in the underlying MAS. (2) *Output Correction* includes Self-Refine (Madaan et al., 2023), which iteratively improves outputs through self-feedback, and AgentDropoutV2 (Wang et al., 2026b), which rectifies erroneous agent outputs or rejects those that remain unreliable. (3) *Structural Adaptation* includes DyLAN (Liu et al., 2023), which combines agent-team selection with dynamic collaboration, and AgentPrune (Zhang et al., 2025b), which prunes spatial and temporal communication dependencies. Detailed baseline descriptions are provided in Appendix C.2.2 and implementation details in Appendix C.3.

4.2.2 RESULTS AND ANALYSIS

Table 2 shows that MAS-Gate consistently outperforms prompted LLM Editors, output-correction methods, and structural-adaptation methods across all six benchmarks, achieving an average score of 47.87 compared with 44.16 for the strongest baseline. Specifically, under the same Editor protocol, the trained Qwen3-8B Editor substantially outperforms the frozen prompted DeepSeek-V4-Flash and GPT-5.4-Mini Editors. This comparison suggests that effective intervention in an ongoing MAS requires more than general reasoning ability and benefits from execution-level feedback that teaches the Editor when and how to adjust the workflow. Furthermore, the proposed online intervention setting outperforms design-time orchestration alone (Table 1) on all six shared benchmarks, highlighting the value of extending MAS optimization into execution, where emerging evidence enables targeted adjustments unavailable at design time. Together, these results support MAS-Gate quantification as a useful supervisory signal for learning this intervention capability, enabling a compact Editor to make effective runtime decisions without access to quantification scores at inference time.

Table 2: Performance comparison of MAS-Gate against baselines on runtime intervention. ALFW, SciW, BCP, and GPQA denote ALFWorld, SciWorld, BrowseComp-Plus, and GPQA-Diamond, respectively. **Bold** indicates the best result.

Category	Method	Train: ALFW		Train: HotpotQA		Train: DeepScaler		Avg
		ALFW	SciW_{OOT}	HotpotQA	BCP_{OOT}	AIME	GPQA_{OOT}	
LLM Editors	DeepSeek-V4-Flash	34.94	7.33	62.00	11.31	31.58	54.04	33.53
	GPT-5.4-Mini	37.50	5.67	57.50	9.52	35.09	59.09	34.06
Output Correction	Self-Refine	48.40	13.67	58.00	5.36	38.60	55.05	36.51
	AgentDropoutV2	53.21	17.00	69.00	9.52	42.11	63.64	42.41
Structural Adaptation	DyLAN	55.77	16.67	72.50	12.50	43.86	63.64	44.16
	AgentPrune	51.28	9.34	70.00	10.12	31.58	61.11	38.91
MAS-Gate	Editor	**59.62**	**22.33**	**75.50**	**15.48**	**49.12**	**65.15**	**47.87**

4.3 Post-Hoc Attribution

4.3.1 Experimental Settings

Risk Score for Error Attribution. For post-hoc error attribution, we represent each recorded trajectory as a chronological chain connecting consecutive steps. We use the negative part of the shared node impact as a risk score:

$$s_i^{\text{risk}} = \max(0, -I_i) = \max(0, -U_i)(1 + K_i). \tag{12}$$

Negating I_i converts adverse impacts into positive risk scores, so that steps with more negative impacts receive higher priority for attribution. Clipping at zero assigns zero risk to non-negative impacts, retaining only the component that indicates reduced support for the reference answer. We rank steps by this score for step-level attribution. For agent-level attribution, we use the maximum risk over an agent's eligible steps, so that a high-risk contribution is not averaged away by its other steps. The resulting top-k ranking is provided to an LLM localizer as auxiliary evidence.

Benchmarks. We evaluate two attribution tasks, i.e., step-level localization on Who&When (Zhang et al., 2025d) and agent-level localization on the single-target test subset of AEGIS (Kong et al., 2026). Who&When provides annotated error steps in failed executions, whereas AEGIS identifies agents subjected to controlled error injection. We report exact step-level and agent-level accuracy for each benchmark. Further benchmark details are provided in Appendix C.1.

Baselines. We compare Direct LLM localization with its MAS-Gate-guided counterpart using the same complete execution trajectory and reference task answer. MAS-Gate additionally supplies a risk-based ranking of candidate steps or agents as auxiliary evidence. The comparison covers five localizers from the Qwen (Team, 2025), Mistral (Jiang et al., 2023), and Llama (Dubey et al., 2024) families. Appendix C.2.3 and C.3.3 provide the baseline definitions and implementation details.

4.3.2 Results and Analysis

Table 3 shows that MAS-Gate improves localization for four of five models on Who&When and all five on AEGIS, supporting its applicability across model families and attribution granularities. Notably, the largest gain on each benchmark occurs for the localizer with the lowest Direct LLM accuracy: Llama-3.1-8B gains 5.43% on Who&When, and Qwen3.5-4B gains 5.84% on AEGIS. This pattern suggests that risk-based guidance can help localizers who struggle to identify decisive evidence within a complete trajectory. Because both settings retain the same trajectory and reference answer, the gains highlight the value of explicitly prioritizing suspicious contributions within the available evidence. While Qwen3.6-27B shows a slight decline on Who&When, MAS-Gate improves accuracy in all other model–benchmark pairs. This broad pattern supports the effectiveness of MAS-Gate quantification in guiding failure attribution across different LLMs and evaluation settings. The ablations in Appendix D.2.1 further support this finding: risk-ranked candidates out-

Table 3: Performance comparison of MAS-Gate against Direct LLM on post-hoc error attribution. We report step-level accuracy on Who&When and agent-level accuracy on the single-target AEGIS subset (%). Δ denotes the change over Direct LLM in percentage points. **Bold** indicates the better result across pairs.

Localizer	Who&When ($k = 8$)			AEGIS ($k = 4$)		
	Direct LLM	+ MAS-Gate	Δ	Direct LLM	+ MAS-Gate	Δ
Qwen3-30B-A3B	33.70	**37.50**	+3.80	43.07	**43.80**	+0.73
Qwen3.5-4B	34.24	**35.87**	+1.63	34.31	**40.15**	+5.84
Qwen3.6-27B	**40.76**	39.67	−1.09	52.55	**54.01**	+1.46
Mistral-Small-3.2-24B	30.98	**32.07**	+1.09	50.36	**51.82**	+1.46
Llama-3.1-8B	21.20	**26.63**	+5.43	38.69	**40.15**	+1.46

perform both randomly selected candidates and the same candidates in shuffled order, indicating that both candidate selection and prioritization contribute to effective attribution.

Figure 2 reveals two patterns in the effect of candidate count on Who&When. First, MAS-Gate provides consistent benefits across candidate budgets: both localizers outperform Direct LLM for every tested k from 2 to 10. The gains at $k = 2$ further show that even a small set of prioritized steps can provide useful guidance within the complete trajectory. Second, the additional benefit of expanding this set depends on the localizer. Qwen3-30B-A3B reaches a plateau at $k = 8$, whereas Llama-3.1-8B continues to improve from $k = 8$ to $k = 10$, despite fluctuations at smaller values. Thus, MAS-Gate remains effective across the tested range, but increasing the breadth of its guidance yields different returns for different models. Together, the improvements across localizers, the ranking ablations, and the sensitivity analysis support MAS-Gate quantification as an effective basis for prioritizing failure-relevant evidence and improving LLM-based post-hoc attribution.

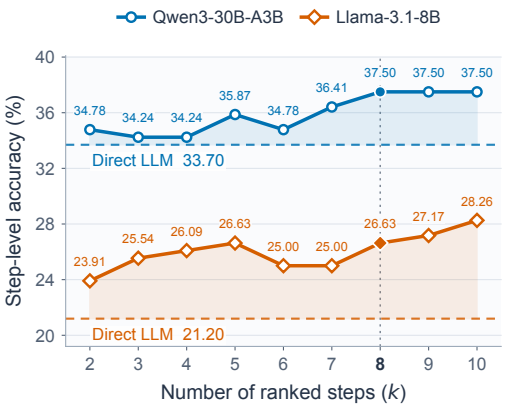


Figure 2: Effect of k on Who&When. Solid lines show MAS-Gate-guided localization and dashed lines show Direct LLM baselines.

5 Conclusion

In this work, we introduce the MAS lifecycle as an execution-centered framework that unifies existing fragmented research on design-time orchestration, runtime intervention, and post-hoc attribution. We then propose MAS-Gate, a unified node-centric quantification that combines within-node utility and inter-node strength to provide structured feedback across all three phases. Experiments demonstrate improved orchestration and generalization, effective trained runtime intervention, and gains in assisted attribution across most evaluated settings, supporting the use of a shared quantification mechanism for these distinct objectives. The runtime results further highlight the benefits of adapting workflows to evidence that emerges during execution. Overall, MAS-Gate demonstrates the feasibility of a common quantification across the lifecycle. However, the three phases are currently optimized independently. Future work could explore how feedback from each phase can inform the others, leading to a more integrated MAS lifecycle. For example, the Orchestrator could learn from execution outcomes after Editor intervention, while the Editor adapts to recurring weaknesses in the generated workflows, allowing the two policies to improve together.

Ethics Statement

This work studies MAS orchestration, runtime intervention, and failure attribution in controlled research benchmarks. Our experiments use established datasets and model-generated execution traces, without recruiting human participants or collecting new personal or sensitive data. The evaluation concerns reasoning, question answering, and interactive tasks rather than decisions about individuals or demographic groups. We use all datasets in accordance with their intended research settings and licenses, and report our methodology and experimental results with an emphasis on transparency and reproducibility

Reproducibility Statement

Our reinforcement-learning implementation uses `verl`. Appendix B presents the MAS-Gate quantification procedure, and Appendix C documents benchmarks, baselines, model configurations, training hyperparameters, and hardware. Appendix F provides prompts and output formats for the Orchestrator, Editor, and LLM-based evaluation components. We provide our implementation code in the supplementary material to facilitate reproduction of the experimental results across all three lifecycle phases.

LLM Usage

LLM assistants were used during manuscript preparation to improve grammar, clarity, organization, and LaTeX formatting. All assisted text is manually reviewed by the authors, and the authors remain responsible for the final content.

References

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

Tiejin Chen, Huaiyuan Yao, Jia Chen, Evangelos E Papalexakis, and Hua Wei. Every response counts: Quantifying uncertainty of llm-based multi-agent systems through tensor decomposition. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16204–16218, 2026a.

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.

Xudong Chen, Yixin Liu, Hua Wei, and Kaize Ding. Lemon: Learning executable multi-agent orchestration via counterfactual reinforcement learning. *arXiv preprint arXiv:2605.14483*, 2026b.

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

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

Shengda Fan, Xin Cong, Yuepeng Fu, Zhong Zhang, Shuyan Zhang, Yuanwei Liu, Yesai Wu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Workflowllm: Enhancing workflow orchestration capability of large language models. In *International Conference on Learning Representations*, volume 2025, pp. 24498–24525, 2025.

Yu Ge, Linna Xie, Zhong Li, Yu Pei, and Tian Zhang. Spectrum-based failure attribution for multi-agent systems. *Proceedings of the ACM on Software Engineering*, 3(FSE):1888–1909, 2026.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

Jiayi He, Hehai Lin, Qingyun Wang, Yi R Fung, and Heng Ji. Self-correction is more than refinement: A learning framework for visual and language reasoning tasks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6405–6421, 2025.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Hui Su, and Xunliang Cai. Vitabench: Benchmarking llm agents with versatile interactive tasks in real-world applications. In *International Conference on Learning Representations*, volume 2026, pp. 91654–91681, 2026.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

Weiquan Huang, Zixuan Wang, Hehai Lin, Sudong Wang, Bo Xu, Qian Li, Beier Zhu, Linyi Yang, and Chengwei Qin. Ama: Adaptive memory via multi-agent collaboration. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 3099–3120, 2026.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL `https://arxiv.org/abs/2310.06825`.

Prathyusha Jwalapuram, Hehai Lin, Chuyuan Li, Fangkai Jiao, Sudong Wang, Yifei Ming, Zixuan Ke, Chengwei Qin, Giuseppe Carenini, and Shafiq Joty. The illusion of multi-agent advantage. *arXiv preprint arXiv:2606.13003*, 2026.

Zixuan Ke, Fangkai Jiao, Yifei Ming, Xuan-Phi Nguyen, Austin Xu, Do Xuan Long, Minzhi Li, Chengwei Qin, Peifeng Wang, Silvio Savarese, et al. A survey of frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems. *arXiv preprint arXiv:2504.09037*, 2025a.

Zixuan Ke, Austin Xu, Yifei Ming, Xuan-Phi Nguyen, Ryan Chin, Caiming Xiong, and Shafiq Joty. Mas-zero: Designing multi-agent systems with zero supervision. *arXiv preprint arXiv:2505.14996*, 2025b.

Zixuan Ke, Yifei Ming, Austin Xu, Ryan Chin, Xuan-Phi Nguyen, Prathyusha Jwalapuram, Jiayu Wang, Semih Yavuz, Caiming Xiong, and Shafiq Joty. Mas-orchestra: Understanding and improving multi-agent reasoning through holistic orchestration and controlled benchmarks. *arXiv preprint arXiv:2601.14652*, 2026.

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

Fanqi Kong, Ruijie Zhang, Huaxiao Yin, Guibin Zhang, Xiaofei Zhang, Ziang Chen, Zhaowei Zhang, Xiaoyuan Zhang, Song-Chun Zhu, and Xue Feng. Aegis: Automated error generation and attribution for multi-agent systems. In *International Conference on Learning Representations*, volume 2026, pp. 52994–53028, 2026.

Moritz Laurer, Wouter van Atteveldt, Andreu Casas, and Kasper Welbers. Building Efficient Universal Classifiers with Natural Language Inference, December 2023. URL `http://arxiv.org/abs/2312.17543`. arXiv:2312.17543 [cs].

Jisoo Lee, Raeyoung Chang, Dongwook Kwon, Harmanpreet Singh, and Nikhil Verma. Gemmas: Graph-based evaluation metrics for multi agent systems. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 1522–1532, 2025.

Shiyuan Li, Yixin Liu, Yu Zheng, Mei Li, Quoc Viet Hung Nguyen, and Shirui Pan. Ofa-mas: One-for-all multi-agent system topology design based on mixture-of-experts graph generative models. In *Proceedings of the ACM Web Conference 2026*, pp. 1333–1344, 2026.

Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9, 2024.

Hehai Lin, Shilei Cao, Sudong Wang, Haotian Wu, Minzhi Li, Linyi Yang, Juepeng Zheng, and Chengwei Qin. Interactive learning for llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 6085–6108, 2026a.

Hehai Lin, Yu Yan, Zixuan Wang, Bo Xu, Sudong Wang, Weiquan Huang, Ruochen Zhao, Minzhi Li, and Chengwei Qin. Unified-mas: Universally generating domain-specific nodes for empowering automatic multi-agent systems. *arXiv preprint arXiv:2603.21475*, 2026b.

Hehai Lin, Qi Yang, and Chengwei Qin. Skill-mas: Evolving meta-skill for automatic multi-agent systems. *arXiv preprint arXiv:2606.18837*, 2026c.

Junnan Liu, Hongwei Liu, Songyang Zhang, and Kai Chen. Rectifying llm thought from lens of optimization. In *International Conference on Learning Representations*, volume 2026, pp. 42967–43008, 2026.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic llm-powered agent network for task-oriented agent collaboration. *arXiv preprint arXiv:2310.02170*, 2023.

Xingjun Ma, Yifeng Gao, Yixu Wang, Ruofan Wang, Xin Wang, Ye Sun, Yifan Ding, Hengyuan Xu, Yunhao Chen, Yunhan Zhao, et al. Safety at scale: A comprehensive survey of large model and agent safety. *Foundations and Trends in Privacy and Security*, 8(3-4):1–240, 2026.

MAA-Committees. Aime problems and solutions. 2025. URL `https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions`.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

Shihao Qi, Jie Ma, Rui Xing, Wei Guo, Xiao Huang, Zhitao Gao, Jianhao Deng, Jun Liu, Lingling Zhang, Bifan Wei, et al. Beyond individual intelligence: Surveying collaboration, failure attribution, and self-evolution in llm-based multi-agent systems. *arXiv preprint arXiv:2605.14892*, 2026.

Abhishek Rath. Agent drift: Quantifying behavioral degradation in multi-agent llm systems over extended interactions. *arXiv preprint arXiv:2601.04170*, 2026.

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.

Jianhao Ruan, Zhihao Xu, Yiran Peng, Fashen Ren, Zhaoyang Yu, Xinbing Liang, Jinyu Xiang, Yongru Chen, Bang Liu, Chenglin Wu, et al. Aorchestra: Automating sub-agent creation for agentic orchestration. *arXiv preprint arXiv:2602.03786*, 2026.

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

Xu Shen, Yixin Liu, Yiwei Dai, Yili Wang, Rui Miao, Yue Tan, Shirui Pan, and Xin Wang. Understanding the information propagation effects of communication topologies in llm-based multi-agent systems. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 12358–12372, 2025.

Xi Shi, Mengxin Zheng, and Qian Lou. Learning latency-aware orchestration for multi-agent systems. *arXiv preprint arXiv:2607.13359*, 2026.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, et al. Toolorchestra: Elevating intelligence via efficient model and tool orchestration. *arXiv preprint arXiv:2511.21689*, 2025.

Sijun Tan, Michael Luo, Justin Wong, Colin Cai, Xiaoxiang Shi, William Yuan Tang, Manan Roongta, Tianjun Zhang, Li Erran Li, Raluca Ada Popa, et al. Deepscaler: Effective rl scaling of reasoning models via iterative context lengthening. 2025.

Qwen Team. Qwen3 technical report, 2025. URL `https://arxiv.org/abs/2505.09388`.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D Nguyen. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*, 2025.

King Yeung Tsang, Zihao Zhao, Vishal Venkataramani, Haizhou Shi, Zixuan Ke, Semih Yavuz, Shafiq Joty, and Hao Wang. Reward modeling for multi-agent orchestration. *arXiv preprint arXiv:2606.13598*, 2026.

Kun Wang, Guibin Zhang, ManKit Ye, Xinyu Deng, Dongxia Wang, Xiaobin Hu, Jinyang Guo, Yang Liu, and Yufei Guo. Mas2: Self-generative, self-configuring, self-rectifying multi-agent systems. In *International Conference on Learning Representations*, volume 2026, pp. 113586–113613, 2026a.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022a.

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

Yutong Wang, Siyuan Xiong, Xuebo Liu, Wenkang Zhou, Liang Ding, Miao Zhang, and Min Zhang. AgentDropoutV2: Optimizing information flow in multi-agent systems via test-time rectify-or-reject pruning. *arXiv preprint arXiv:2602.23258*, 2026b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Shanglin Wu and Kai Shu. Memory in llm-based multi-agent systems: Mechanisms, challenges, and collective intelligence. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 167–184. Springer, 2026.

Yufei Xia, Anjun Gao, Yueyang Quan, Zhuqing Liu, and Minghong Fang. Who broke the system? failure localization in llm-based multi-agent systems. *arXiv preprint arXiv:2607.07989*, 2026.

Yingxuan Yang, Chengrui Qu, Muning Wen, Laixi Shi, Ying Wen, Weinan Zhang, Adam Wierman, and Shangding Gu. Understanding agent scaling in llm-based multi-agent systems via diversity. *arXiv preprint arXiv:2602.03794*, 2026.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Rui Ye, Shuo Tang, Rui Ge, Yaxin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. Mas-gpt: Training llms to build llm-based multi-agent systems. *arXiv preprint arXiv:2503.03686*, 2025.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15549–15572, 2025.

Chenchen Zhang. Reinforcement learning for llm-based multi-agent systems through orchestration traces. *arXiv preprint arXiv:2605.02801*, 2026.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*, 2024.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025a.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for LLM-based multi-agent systems. In *International Conference on Learning Representations*, volume 2025, pp. 75389–75428, 2025b.

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, 2025c.

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*, 2025d.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, et al. Where llm agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*, 2025.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*, 2024.

A Description of Appendix

This appendix provides methodological details and additional experimental evidence for MAS-Gate. Appendix B presents the shared node-centric quantification procedure and its use across the three lifecycle phases. Appendix C documents benchmark statistics, baseline configurations, and implementation details for each experiment. Appendix D examines node-impact composition, the contribution of the design-time dense reward, and the roles of candidate selection and ranking in failure attribution. Appendix E illustrates workflow construction, runtime intervention, and post-hoc attribution through concrete execution traces. Finally, Appendix F provides the prompts and output formats used for orchestration, editing, failure localization, and benchmark evaluation.

B Pseudocode of MAS-Gate

We provide the pseudocode of MAS-Gate here, which summarizes the common computation performed on an execution trace.

Algorithm 1 MAS-Gate node-centric quantification and lifecycle use

Require: Task x, reference y, execution DAG $\mathcal{G} = (V, E)$, outputs $\{o_i\}_{i \in V}$, scoring LLM p_ϕ, NLI model p_{NLI}, phase s

Ensure: Node records $\mathcal{S}$, edge scores $\{R_{ij}\}_{(i,j)\in E}$, and phase-specific signal

$J_0 \leftarrow J_\phi(x, y)$
for i in a topological ordering of V **do**
  $C_i^- \leftarrow \mathrm{Concat}(x, \{o_a : a \in \mathrm{Anc}(i)\})$
  $C_i^+ \leftarrow \mathrm{Concat}(x, \{o_a : a \in \mathrm{Anc}(i)\}, o_i)$
  $U_i \leftarrow \tanh\big((J_\phi(C_i^+, y) - J_\phi(C_i^-, y))/\,|J_0|\big)$
end for
for $(i, j) \in E$ **do**
  $R_{ij} \leftarrow p_{\mathrm{NLI}}(\mathrm{entailment} \mid o_i, o_j)$
end for
$D_{\max} \leftarrow \max_{v \in V} |\,\mathrm{Desc}(v)|$
$D_{\max} > 0$ since the execution graph contains at least one dependency edge.
for $j \in V$ **do**
  $G_j \leftarrow |\,\mathrm{Desc}(j)|/D_{\max}$
end for
for $i \in V$ **do**
  $K_i \leftarrow \sum_{j \in \mathrm{Ch}(i)} R_{ij}(1 + G_j)$
  $I_i \leftarrow U_i(1 + K_i)$
  Add (i, U_i, K_i, I_i) and outgoing R_{ij} values to $\mathcal{S}$
end for
if $s =$ Design **then**
  return $\mathcal{S}$ and $|V|^{-1} \sum_{i \in V} I_i + c(\tau)$
else if $s =$ Runtime **then**
  Cache reference τ^0 and mean impact $\bar{I}(\tau^0) \leftarrow |V|^{-1} \sum_i I_i$
  Sample π_θ on public execution prefixes; allow at most one non-Keep edit
  Set $r_t \leftarrow 0$ for legal Keep decisions; $r_t \leftarrow -0.25$ for invalid actions
  if a legal edit a_t is selected **then**
    Apply a_t, complete and quantify τ^{a_t}, then set $r_t \leftarrow r(a_t)$ via Eq. 11
  end if
  return $\mathcal{S}$ and turn-local rewards $\{r_t\}$ for GRPO
else
  Compute step risks $s_i^{\mathrm{risk}} \leftarrow \max(0, -I_i)$
  For agent-level attribution, aggregate $s_a^{\mathrm{risk}} \leftarrow \max_{i \in \mathcal{V}_a^{\mathrm{elig}}} s_i^{\mathrm{risk}}$
  Rank eligible steps or agents by decreasing risk and retain the top-k as $\mathcal{R}_k$
  return $\mathcal{S}$ and the predicted error step or agent from the localizer given x, y, full τ, and $\mathcal{R}_k$
end if

Table 4: Benchmark roles and split statistics. OOT indicates that the benchmark is never used to update or select the Orchestra trained on the paired source task.

Task Family	Benchmark	Role	Train size	Test size	Task characteristic
Interactive environments	ALFWorld	Train / In-task test	4032	312	Language-grounded household interaction
	SciWorld	OOT test only	-	300	Interactive scientific environments
	VitaBench	OOT test only	-	100	Multi-domain tool-use interaction
Multi-hop QA	HotpotQA	Train / in-task test	10000	200	Multi-document evidence composition
	BrowseComp-Plus	OOT test only	-	168	Hard information aggregation
General reasoning	DeepScaleR	Train only	10000	-	Complex mathematical reasoning
	AIME	In-task test only	-	57	Competition mathematics (Year 24&25)
	GPQA-Diamond	OOT test only	-	198	Graduate-level science questions
Failure attribution	Who&When	Evaluation only	-	184	Exact step-level attribution
	AEGIS	Evaluation only	-	137	Single-target agent-level attribution

C Experimental Details

C.1 Statistics of Benchmarks

Table 4 summarizes how each benchmark is used across the experiments. ALFWorld, HotpotQA, and DeepScaleR serve as the three source benchmarks for training the design-time Orchestra on interactive tasks, multi-hop question answering, and general reasoning, respectively. Their associated in-task evaluations measure performance within the capability family represented in training. SciWorld, VitaBench, BrowseComp-Plus, and GPQA-Diamond are held-out out-of-task (OOT) benchmarks, which are used to measure the transfer performance to related but distinct task distributions.

Interaction environments. ALFWorld aligns abstract text environments with embodied household tasks (Shridhar et al., 2020). Its episodes require an agent to interpret a high-level language goal, plan over a long horizon, manipulate household objects, and adjust to the current layout and state. **SciWorld** places the agent in a text-based scientific environment where it must inspect objects, operate instruments, and conduct experiments through ordered actions before determining the scientific result (Wang et al., 2022a). **VitaBench** evaluates multi-turn tool-using agents in simulated service settings, including food delivery, in-store consumption, and online travel (He et al., 2026). Its evolving user goals, dependent tool calls, and changing service states probe practical decision-making and recovery over extended interactions.

Multi-hop QA. HotpotQA is a Wikipedia-based multi-hop QA benchmark whose questions require evidence to be gathered and combined across multiple documents (Yang et al., 2018). **BrowseComp-Plus** evaluates deep-research agents on difficult information-seeking questions over a fixed, curated corpus (Chen et al., 2025). The fixed corpus avoids dependence on the changing open web while preserving the need for iterative retrieval, source comparison, and evidence synthesis, making the benchmark appropriate for testing transfer from standard multi-hop QA to broader research-oriented reasoning.

General reasoning. The **DeepScaleR** training set contains competition-style mathematical problems collected for reinforcement-learning-based reasoning training (Tan et al., 2025). The problems demand multi-step derivations but retain verifiable final answers, which provides a reliable outcome signal for optimization. **AIME** consists of challenging competition mathematics problems spanning

algebra, geometry, combinatorics, and number theory (MAA-Committees, 2025). **GPQA-Diamond** comprises expert-written graduate-level questions in biology, physics, and chemistry (Rein et al., 2023). It tests scientific reasoning and specialized knowledge outside the mathematical distribution used for training, thereby serving as an OOT transfer evaluation.

Failure attribution. For **Who&When,** we use all 184 publicly released failure trajectories, comprising 126 algorithm-generated and 58 hand-crafted examples (Zhang et al., 2025d). Each example contains the task query, reference answer, complete execution history, and an annotated error agent and step. We retain the released zero-based step indices and evaluate whether the predicted step exactly matches the annotated error step. Algorithm-generated trajectories contain 5-10 steps, with a mean of 8.72, while hand-crafted trajectories contain 5-130 steps, with a mean of 51.60. **AEGIS** constructs failure trajectories through controlled error injection into multi-agent executions (Kong et al., 2026). We use the test split and retain examples whose injection annotations identify exactly one distinct agent, yielding 137 of the 600 released test examples. The subset covers GAIA, GSM8K, SciBench, HumanEval, and MMLU-Pro, with trajectory lengths ranging from 3 to 43 steps and a mean of 13.83. The selection criterion concerns the number of distinct agents, rather than the number of injection annotations. We evaluate exact agent-level accuracy against the unique target agent. Model inputs retain the task query, reference answer, observed final output, and complete conversation history, but exclude injection indicators and error annotations.

C.2 Baselines

C.2.1 Design-time orchestration

General inference methods. CoT prompts the frozen LLM to produce intermediate reasoning before the final answer in one-shot (Wei et al., 2022). **CoT-SC** samples multiple reasoning trajectories and selects the answer with the highest consistency (Wang et al., 2022b). These methods do not construct an explicit multi-agent workflow and provide single-agent references for judging the value of learned coordination.

Inference-time MAS methods. AFlow searches over code-represented agentic programs and improves candidate workflows using Monte Carlo Tree search (Zhang et al., 2025c). **AOrchestra** instantiates task-specific sub-agents by configuring their instructions, context, tools, and models at inference time (Ruan et al., 2026). **Skill-MAS** stores orchestration experience in an evolvable meta-skill and uses trajectory sampling and reflection to construct workflows without updating model parameters (Lin et al., 2026c). These methods adapt the workflow at inference time rather than training an orchestrator.

Training-time MAS methods. MAS-GPT learns to generate executable, query-adaptive multi-agent programs (Ye et al., 2025). **ToolOrchestra** trains a lightweight orchestrator to coordinate models and tools under outcome-, efficiency-, and user-preference-aware rewards (Su et al., 2025). **MAS-Orchestra** formulates holistic MAS construction as function-calling reinforcement learning in which the orchestrator generates the complete organization (Ke et al., 2026). These are the closest design-time comparisons. These methods use distinct training objectives and orchestration formulations.

C.2.2 Runtime intervention

LLM Editors. DeepSeek-V4-Flash and **GPT-5.4-Mini** serve as prompted editing controllers, while the underlying main-agent and sub-agent components are retained. Their role is to select runtime interventions from the observed execution state rather than directly replace the task-solving agents. This group compares general-purpose LLM prompting with an Editor specifically trained for runtime intervention using MAS-Gate feedback.

Output Correction. Self-Refine iteratively improves a candidate output by having the same LLM generate feedback and revise its answer accordingly, without additional parameter training (Madaan et al., 2023; He et al., 2025). **AgentDropoutV2** intercepts intermediate agent outputs and retrieves failure indicators to guide error correction; outputs that remain unreliable are rejected to limit downstream error propagation (Wang et al., 2026b). These methods focus on correcting or filtering generated content, providing output-level comparisons to our Editor, which can additionally select re-execution and workflow interventions.

Table 5: Key hyperparameters used in our experiments.

Hyperparameter	Value
RL framework	verl
GPU	8×H100
Train batch size	256
Rollout group size	16
PPO mini-batch size	256
PPO micro-batch size per GPU	2
Log-probability micro-batch size per GPU	4
Maximum prompt length	15,000
Maximum response length	8,192
Actor learning rate	1×10^{-6}
PPO epochs	1
PPO clipping ratio	0.2
KL coefficient	0.001
Gradient clipping norm	1.0
Maximum concurrency	1,024

Structural Adaptation. DyLAN selects a task-oriented agent team based on agent contributions and supports dynamic collaboration during task solving (Liu et al., 2023). **AgentPrune** optimizes communication topology through one-shot pruning of spatial and temporal message-passing dependencies, reducing redundant information exchange (Zhang et al., 2025b). These methods adapt participating agents or communication structure. They provide structure-level comparisons to MAS-Gate's learned selection among both content-level and workflow-level interventions.

C.2.3 Post-hoc attribution

Direct LLM. Our baseline follows the All-at-Once formulation in Who&When (Zhang et al., 2025d), with the reference task answer included in the input. Given the task query, complete trajectory, and reference answer, the localizer predicts an error step and its agent on Who&When, or a single responsible agent on AEGIS, together with a brief explanation. MAS-Gate-guided inference uses the same localizer, task input, and output format, and appends the risk-based candidate ranking.

C.3 Implementation Details

C.3.1 Design-time orchestration

Table 5 summarizes key hyperparameters used in our reinforcement learning. Qwen3-8B (Team, 2025) is the only trainable model and serves as the design-time Orchestra and runtime Editor. Qwen3-30B-A3B-Instruct-2507 (Team, 2025) remains frozen while executing sub-agent nodes and providing the teacher-forced likelihoods used by the within-node utility. The frozen bge-m3-zeroshot-v2.0 (Laurer et al., 2023) service provides the entailment probability used for inter-node propagation. For ALFWorld, we construct the reference label y as an abstract representation of the final state specified by the training example and use it for within-node utility quantification.

C.3.2 Runtime intervention

Runtime Editor protocol. We freeze the original Qwen3-8B checkpoint as the Orchestrator throughout Editor training and evaluation, independently of the orchestration training reported in Table 1. MAS-Gate supplies privileged training feedback through paired rewards evaluated on completed edited and unedited trajectories. Editor decisions are specified to condition on the task, graph, and observed execution prefix. At test time, the Editor receives neither the reference answer nor node-level quantification scores and makes at most one non-KEEP intervention per execution.

Runtime Editor fine-tuning. We optimize LoRA adapters for the Editor using GRPO, with rank 64, scaling factor $\alpha = 32$, and zero dropout, applied to all linear layers while keeping the base-model weights frozen. Each episode allows at most four additional node executions. In Eq. 11, we

use $\beta = 0.1$ and $\lambda = 1.0$. Action costs are 0.03 for annotation and child guidance, 0.08 for retry, 0.10 for insertion, and 0.12 for backtracking; invalid actions receive a reward of -0.25.

Action Space. The six actions operate on different parts of the execution. KEEP preserves the current output and advances to the next eligible gate. RETRY-CURRENT re-executes the current node using its recorded prompt and context. BACKTRACK-PARENT selects a direct parent and re-executes that parent together with its already executed affected descendants. ANNOTATE-CURRENT preserves the raw current output but appends a bounded diagnosis for downstream nodes. GUIDE-CHILD adds a bounded operational instruction to one unexecuted direct child without introducing a new node, whereas INSERT-SUBTASK inserts one LLM-only verification, bridge, or supplement node between the current node and such a child. The legal set is constructed dynamically from the graph position, execution status, replay safety, and remaining node-call budget.

C.3.3 POST-HOC ATTRIBUTION

We use Qwen3-30B-A3B-Instruct-2507 for utility estimation and bge-m3-zeroshot-v2.0 for inter-node entailment scoring. Each recorded trajectory is represented as a chronological chain, with edges connecting consecutive steps as a proxy for temporal dependencies. The resulting MAS-Gate risk scores are computed once and reused across all localizers. The attribution results therefore characterize MAS-Gate under a temporal chain approximation.

The five localizers are Qwen3-30B-A3B-Instruct-2507, Qwen3.5-4B, Qwen3.6-27B, Mistral-Small-3.2-24B-Instruct-2506, and Llama-3.1-8B-Instruct. All localizers are run with vLLM using a temperature of 0, a maximum output length of 256 tokens, and an 81,920-token context window. Thinking mode is disabled where supported. Direct LLM and MAS-Gate-guided inference use the same inference settings and retain the complete trajectory and reference task answer.

We set $k = 8$ on Who&When and $k = 4$ on AEGIS. For AEGIS, steps are grouped by their recorded agent names, and each agent is ranked by its maximum eligible-step risk. User and System turns are retained as context but excluded from attribution targets. When fewer than k candidates are available, all are included. Guidance consists of ranked step indices on Who&When and ranked agent names on AEGIS; numerical risk scores are not included in the prompts. The prompt templates are provided in AppendixF.

D ADDITIONAL ABLATION EXPERIMENTS

D.1 DESIGN-TIME ORCHESTRATION

The following ablations are organized around the principal claims made by the method rather than around arbitrary parameter sweeps. Each ablation changes one component while holding the training split, sub-agent backbone, total execution budget, and evaluation protocol fixed. Table 6 reports the resulting accuracy on HotpotQA and the out-of-task BrowseComp-Plus benchmark.

D.1.1 ALTERNATIVE NODE-IMPACT COMPOSITION

We compare the default multiplicative impact $I_i = U_i(1 + K_i)$ with the additive alternative $I_i = U_i + (1 + K_i)$. All other components, including the within-node utility, inter-node strength, training data, and evaluation protocol, remain unchanged. As shown in Table 6, the additive variant reaches 66.50 on HotpotQA and 11.90 on BrowseComp-Plus, below the corresponding MAS-Gate results of 69.50 and 13.69. We hypothesize that the multiplicative form better represents the intended interaction between node utility and propagation strength: downstream influence scales the value of the information produced by a node, so the two factors jointly determine its impact. In contrast, the additive form treats local utility and propagation more independently, allowing one component to offset the other and thereby weakening the distinction between information that is locally useful and information that is also effectively transmitted through the workflow.

D.1.2 DESIGN-TIME REWARD ABLATION

This ablation tests whether the design-time Orchestra benefits from the node-centric dense signal beyond terminal correctness. We train two otherwise identical Orchestra policies with the same

Table 6: Ablation results for node-impact composition and design-time reward. BCP denotes the out-of-task BrowseComp-Plus evaluation. All values are accuracy (%).

Training variant	HotpotQA	BCP_{OOT}
Full MAS-Gate	**69.50**	**13.69**
Additive impact	66.50	11.90
Accuracy-only reward	64.50	9.52

initialization, rollout groups, optimization budget, sub-agent backbone, and task mixtures. The first policy uses only the terminal accuracy signal,

$$r_{\text{acc}}(\tau) = c(\tau), \tag{13}$$

where $c(\tau) \in \{0, 1\}$ indicates whether the completed execution produces a correct answer. The second policy uses the composite reward from the main experiment,

$$r_{\text{dense+acc}}(\tau) = \frac{1}{|V|} \sum_{i \in V} I_i + c(\tau). \tag{14}$$

Thus, the comparison changes only the training signal: both policies are prompted to generate the same class of executable workflows and are evaluated with the same terminal accuracy metric. Table 6 shows that the fused reward improves HotpotQA accuracy from 64.50 to 69.50 and BrowseComp-Plus accuracy from 9.52 to 13.69, corresponding to gains of 5.00 and 4.17 percentage points, respectively, or 4.59 points on average. The out-of-task result is particularly informative: accuracy-only training falls below the CoT baseline on BrowseComp-Plus (9.52 versus 11.31 in Table 1), whereas the fused reward surpasses it with 13.69. This reversal suggests that sparse binary correctness can overfit the source-task outcome without adequately distinguishing the quality of intermediate coordination. By incorporating graded node-level contributions, the fused reward provides a richer training signal and yields stronger transfer to workflows required by the unseen task distribution.

D.2 Post-hoc attribution

D.2.1 Error Attribution Guidance Ablation

This ablation examines the contributions of candidate selection and risk-based ordering to error localization. We use Qwen3-30B-A3B on Who&When with $k = 8$, keeping the complete trajectory, reference task answer, and inference settings unchanged. In addition to Direct LLM, we compare three guided variants: randomly sampled candidates, MAS-Gate candidates presented in shuffled order, and MAS-Gate candidates ranked by risk. The guided variants use the same prompt template and differ only in candidate selection or presentation order.

Table 7 shows that random candidate guidance achieves 34.78% accuracy, compared with 33.70% for Direct LLM. Shuffled MAS-Gate candidates achieve 35.33%, while retaining their risk-based ranking yields 37.50%. Since the ranked and shuffled variants use the same candidate steps, this comparison provides evidence that the ordering itself offers useful information for localization. Together, these results suggest that the localizer benefits from both MAS-Gate's selection of suspicious steps and its ranking of these steps by risk.

E Case Study

E.1 Design-Time Orchestration

We select one HotpotQA instance for which an untrained Orchestra generates an MAS that produces an incorrect answer, whereas the Orchestra trained with MAS-Gate produces the correct answer (i.e., Skill-MAS vs. MAS-Gate). In this instance, Skill-MAS returns `None`, whereas MAS-Gate correctly returns **Martin O'Neill**. Skill-MAS spends four of five nodes repeatedly resolving or verifying the easier bridge entity, leaving the harder successor relation to one terminal node. Contrastively, trained

Table 7: Ablation of risk-based guidance on Who&When with Qwen3-30B-A3B as the localizer and $k = 8$. Random Top-8 provides randomly selected candidate steps, whereas MAS-Gate (shuffled) presents the MAS-Gate candidates in random order. We report exact step-level accuracy (%).

Method	Accuracy (%)
Direct LLM	33.70
+ Random Top-8	34.78
+ MAS-Gate (shuffled)	35.33
+ MAS-Gate	**37.50**

MAS-Gate resolves the bridge once, then allocates dedicated nodes to successor discovery, timeline verification, and answer synthesis.

Case information

Benchmark. HotpotQA, task ID `5ac175635542994ab5c67d4b`.
Question. "Who replaced the manager of the Aston Villa that began at Leeds United?"
Label. Martin O'Neill.
Skill-MAS's Answer. None. ×
MAS-Gate's Answer. Martin O'Neill. ✓

Skill-MAS Node 1 — `extract_entities`

Role. Extract manager and club candidates explicitly mentioned in retrieved evidence.
Retrieved/used evidence. *David O'Leary*, *Mervyn Day*, *Ian Culverhouse*, *Gérard Houllier*, and Aston Villa season pages.
Response.

```
managers = [David O'Leary, Ian Culverhouse, Gérard
Houllier, Tim Sherwood, Rémi Garde, Eric Black]
clubs = [Aston Villa, Leeds United, Al-Ahli Dubai, West
Ham United, Leyton Orient, Luton Town, Sheffield United,
Carlisle United]
```

Skill-MAS Node 2 — `find_leeds_manager`

Role. Determine which extracted manager began a managerial career at Leeds United.
Direct predecessor. `extract_entities`.
Retrieved/used evidence. *David O'Leary*: managerial career began at Leeds United; *Mervyn Day* and *1996 Football League Cup Final* act as lexical distractors.
Response.

```
David O'Leary
```

Skill-MAS Node 3 — `find_villa_manager`

Role. Check that Leeds-linked candidate also managed Aston Villa.
Direct predecessor. `find_leeds_manager`.
Retrieved/used evidence. *David O'Leary* states that he later managed Aston Villa; *List of Aston Villa F.C. managers* provides club-manager context.
Response.

```
David O'Leary
```

Skill-MAS Node 4 — `verify_relation`

Role. Verify relation between identified Aston Villa manager and Leeds United.
Direct predecessor. `find_villa_manager`.
Retrieved/used evidence. David O'Leary biography plus Aston Villa manager chronology.
Response.

```
David O'Leary was the manager of Aston Villa and had a
prior association with Leeds United.
```

Skill-MAS Node 5 — `identify_replacement` (localized failure)

Role. Identify who replaced the resolved Aston Villa manager and return one supported name.
Direct predecessor. `verify_relation`.
Retrieved/used evidence. Node 5 does receive two relevant pieces of evidence: (i) *David O'Leary* states that his managerial career began at Leeds United and that he later managed Aston Villa; and (ii) *2010–11 Aston Villa F.C. season* identifies Martin O'Neill as Aston Villa's previous manager before Gérard Houllier. Thus, candidate **Martin O'Neill** is already present in the evidence. Still, the relation between the two pieces is indirect: retrieved snippets do not explicitly state that Martin O'Neill replaced David O'Leary.
Expectation. Resolve exact successor predicate $\textsc{ReplacedAt}(x, \text{David O'Leary}, \text{Aston Villa})$, retrieve or cross-check direct chronology evidence, and emit one name.
Response.

```
Based on the verified relation that David O'Leary was the
manager of Aston Villa and had a prior association with
Leeds United, the evidence does not explicitly provide
the name of the person who replaced him at Aston Villa.
Therefore, no supported candidate can be determined from
the given evidence.
Final answer:  None
```

Failure cause. Failure is best characterized as **evidence integration and completion failure**, not complete absence of relevant evidence. Node 5 sees David O'Leary bridge fact and Martin O'Neill timeline cue, but fails to connect them into candidate chain `David O'Leary` $\rightarrow$ `Martin O'Neill` and then verify missing replacement edge. Once evidence is indirect, the node has no recovery stage for focused retrieval or independent verification and immediately returns `None`. MAS-Gate avoids this failure because **Successor Finder** uses the resolved David O'Leary entity to retrieve explicit replacement documents, while **Timeline Verifier** separately confirms the same edge.

MAS-Gate Node 1 — Bridge Resolver (`agent_0`)

Role. Resolve the Aston Villa manager described through Leeds United.
Retrieved/used evidence. The decisive document is *David O'Leary* (`e4b9...ef5`), which states that his managerial career began at Leeds United and that he later managed Aston Villa. Retrieved distractors include *Mervyn Day*, *1996 Football League Cup Final*, *List of Aston Villa F.C. managers*, *Andy Marshall*, *Joe Mercer*, *Stiliyan Petrov*, and *Gérard Houllier*.
Response.

```
The Aston Villa manager who previously played at Leeds
United is David O'Leary.
```

MAS-Gate Node 2 — Successor Finder (`agent_1`)

Role. Find replacement for the identified Aston Villa manager.
Direct predecessor. `agent_0`; upstream output fixes bridge entity as **David O'Leary**.
Retrieved/used evidence. First-round retrieval returns *2008–09 Aston Villa F.C. season* (`0544...50a`), which states that Martin O'Neill was in his third season since replacing David O'Leary, and *2009–10 Aston Villa F.C. season* (`b597...79f`), which states that he was in his fourth season since replacing David O'Leary. *David O'Leary* confirms bridge entity, while *List of Aston Villa F.C. managers* provides chronology background.
Response.

```
The replacement of David O'Leary as the manager of Aston
Villa was Martin O'Neill.
```

MAS-Gate Node 3 — Timeline Verifier (`agent_2`)

Role. Verify proposed replacement identity and temporal relation.
Direct predecessor. `agent_1`; upstream candidate is **Martin O'Neill**, linked to predecessor **David O'Leary**.
Retrieved/used evidence. Retrieved titles include *David O'Leary*, *2008–09 Aston Villa F.C. season*, *Mervyn Day*, *O'Leary Manager 2000*, and *1996 Football League Cup Final*. Explicit season evidence directly confirms that Martin O'Neill replaced David O'Leary.
Response.

```
David O'Leary was the manager of Aston Villa before being
replaced.  According to the evidence from ``2008--09
Aston Villa F.C. season,'' Martin O'Neill replaced David
O'Leary as the manager of Aston Villa.
Thus, the manager who replaced David O'Leary at Aston
Villa was Martin O'Neill.
```

MAS-Gate Node 4 — Answer Synthesizer (`agent_3`)

Role. Compose final answer from verified two-hop chain.
Direct predecessor. `agent_2`; upstream output explicitly verifies **Martin O'Neill** as replacement of **David O'Leary**.
Retrieved/used evidence. No additional retrieval is needed because both required predicates have been closed upstream.
Response.

```
The manager of Aston Villa who previously played at Leeds
United was David O'Leary.  He was replaced by Martin
O'Neill.
```

E.2 Runtime Intervention

We illustrate runtime intervention on a HotpotQA instance using the same initial three-node workflow and observed prefix. Without intervention, the downstream continuation returns an incorrect financing provider. A single RETRY-CURRENT action produces a more informative retrieval output, allowing the unchanged downstream workflow to answer correctly under semantic evaluation.

Case information

Benchmark. HotpotQA, task ID `5ab5cd0d554299488d4d9a30`.
Question. "The not-for-profit company which owns a Mark VI monorail was assisted in bond financing by whom?"
Gold answer. The State of Nevada.
No-intervention answer. Federal Transit Administration (FTA).
Edited answer. The State of Nevada provided bond financing assistance to the Las Vegas Monorail Company.
Evaluation. Semantic correctness changes from 0 to 1; this is not a string-level exact-match comparison.

Workflow and observed handoff

The workflow contains three nodes: `agent_0` retrieves the company using the BM25 corpus, `agent_1` identifies its bond-financing provider from the retrieved information, and `agent_2` synthesizes the answer. The initial output of `agent_0` includes:

```
The Las Vegas Monorail Company is a not-for-profit
corporation that owns and operates the Las Vegas
Monorail, which uses Mark VI monorail trains.
```

The company is correctly identified, but the handoff does not supply the financing provider required downstream. The Editor is queried at this point, before either downstream node executes.

Recorded Editor action

The Editor emits:

```
{"action":"RETRY_CURRENT","target":"agent_0","payload":{}}
```

The runtime re-executes the current node with its saved prompt and context. The graph remains unchanged. The action record contains no explicit natural-language diagnosis and does not provide a replacement answer.

No-intervention continuation

With the original handoff, `agent_1` produces:

```
The entity that provided bond financing assistance to
the Las Vegas Monorail Company was the Federal Transit
Administration (FTA).
```

The final node repeats `Federal Transit Administration (FTA)`. This incorrect entity appears after the intervention point and was not visible to the Editor when it selected its action.

Edited continuation

After the retry, `agent_0` supplies the missing financing relation:

```
The State of Nevada provided bond financing assistance
for the project, though no public funds were used in
construction.
```

Both downstream nodes subsequently identify the State of Nevada as the financing provider. The final answer semantically matches the gold answer. This example illustrates how a targeted retry can improve the information available to downstream agents without regenerating the workflow or having the Editor directly write the answer.

E.3 Post-hoc Attribution

We illustrate post-hoc attribution on a Who&When trajectory in which incorrect sale-price information is repeatedly accepted by downstream agents. Given the same complete trajectory and reference task answer, Direct LLM attributes the failure to a later validation step, whereas the MAS-Gate-guided localizer identifies the annotated decisive error where the incorrect information first enters the trajectory.

Case information

Benchmark. Who&When, trajectory `Algorithm-Generated/4`.
Task. Compare the 2022 sale prices of 2072 Akaikai Loop and 2017 Komo Mai Drive in Pearl City, Hawaii, and return the higher price without commas or decimal places.
Reference answer. `900000`.
Recorded final answer. `950000`.
Annotated decisive error. Step 2, `HawaiiRealEstate_Expert`.
Localizer. Qwen3-30B-A3B, with $k = 8$ for MAS-Gate guidance.
Indexing and evaluation. Steps are zero-based. Direct LLM selects Step 4; MAS-Gate-guided attribution selects Step 2, matching the released annotation. Both predictions concern the same failed execution.

Introduction and subsequent repetition of the error

At Step 2, `HawaiiRealEstate_Expert` first supplies the sale prices. The recorded price claims are excerpted below, with formatting simplified:

```
2072 Akaikai Loop:
Sold for $850,000
2017 Komo Mai Drive:
Sold for $950,000
```

At Steps 3–4, `Validation_Expert` repeats these prices, compares them, and accepts `950000` as the answer. Its Step 4 response concludes:

```
Since all validation checks are met and the conclusion
aligns with the provided data, we can consider this task
complete.
```

At Step 6, `Data_Analysis_Expert` again reports `950000`. The released annotation assigns the decisive error to Step 2, explaining that the incorrect price propagates through subsequent steps to the final output.

Direct LLM attribution

Direct LLM selects **Step 4**, `Validation_Expert`, and gives the following explanation:

```
Incorrectly accepted and reported $950,000 as the sale
price without verifying against the correct answer
of 900000, indicating a failure to validate against
authoritative data.
```

This diagnosis focuses on the downstream acceptance of the incorrect price, rather than the earlier step identified by the annotation.

MAS-Gate-guided attribution

MAS-Gate assigns its highest risk score to **Step 2**. This trajectory contains eight recorded steps, so the top-8 guidance includes every step and adds their risk-based ordering to the prompt. With this guidance, the same localizer selects **Step 2**, `HawaiiRealEstate_Expert`, and explains:

```
Provided incorrect sale prices for the homes, stating
2072 Akaikai Loop sold for $850,000 and 2017 Komo Mai
Drive for $950,000, when the correct answer is 900000 for
the higher-priced home.
```

The predicted agent and step both match the released annotation. This case illustrates how MAS-Gate guidance can help the localizer identify where erroneous information first enters a trajectory, even when subsequent agents repeatedly endorse it.

F Prompt Details

Figures 3–7 present the five prompts used in our experiments: the Orchestrator prompt in Figure 3, the Editor prompt in Figure 4, the LLM-based error-attribution prompt for Experiment 3 from Figure 5 to Figure 6, and the VitaBench LLM-as-a-judge prompt in Figure 7.

MAS-Gate Prompt for Orchestrator

System_prompt:
You are MAS-Gate's orchestra policy. Design a multi-agent system, not the final answer.
Return XML only. Do not solve the problem directly.
User_prompt:
Dataset: {dataset_name}
Task:
{question}
Output exactly one XML document with this shape:

```
<editor_output>
    <task_decomposition>
        <goal>overall objective</goal>
        <sub_tasks>
            <sub_task idx="0">...</sub_task>
            <sub_task idx="1">...</sub_task>
        </sub_tasks>
    </task_decomposition>
    <agent_specs>
        <agents>
            <agent id="agent_0" is_final="false" execution_mode="llm_only">
                <role>specialized role and instructions</role>
                <sub_task_idx>0</sub_task_idx>
                <dependencies></dependencies>
            </agent>
        </agents>
    </agent_specs>
</editor_output>
```

Figure 3: MAS-Gate XML Orchestrator prompt. The user prompt is assembled dynamically from the task, dataset name, and dataset-specific routing/output constraints.

MAS-Gate Prompt for Editor

System prompt (excerpt):
You are the runtime editor for a multi-agent system. At each gate, inspect only the visible graph prefix and select one legal action.
Decision precedence:
1. First classify whether the current milestone completed its visible role, task, constraints, and (for a final node) the requested global deliverable.
2. If it is clearly INCOMPLETE because the current node omitted or failed its assigned deliverable, `RETRY_CURRENT` has positive expected value and you must select `RETRY_CURRENT` when it is legal; `KEEP` is not valid for that case.
3. If the visible failure instead originates in a parent handoff, select `BACKTRACK_PARENT` when legal.
4. Only after the completion check, compare disruption and extra-call costs among suitable repairs. For a COMPLETE milestone, `KEEP` unless a specific legal repair has positive expected net value. Do not invent an error to justify editing.
[Additional instructions omitted.]
Do not solve the global task yourself. Return exactly one JSON object matching:
`{"action":"ACTION","target":"NODE_OR_NULL","payload":{...}}`
`KEEP` has null target and an empty payload. Use only the legal actions and targets shown in the observation. Never invent hidden outputs or a ground-truth answer.

User prompt (excerpts):
[Milestone-completion checks and the action-selection guide omitted.]
Payload rules:
`ANNOTATE_CURRENT`: `error_type`, `target_span`, `diagnosis`, `downstream_check`. `error_type` must be one of LOGIC, CALCULATION, EVIDENCE, CONSTRAINT, INCOMPLETE, CONFLICT, LOW_CONFIDENCE. Copy `target_span` only from `observed_outputs[current_node]`, never from the question, role, or task. Keep it under 12 words and never write a corrected answer. Maximum lengths are 24 tokens for `target_span`, 48 for `diagnosis`, 24 for `downstream_check`, and 96 total.
`GUIDE_CHILD`: `guidance_type` and `focus`. `guidance_type` is one of VERIFY_UPSTREAM, RECOMPUTE, CHECK_CONSTRAINT, RESOLVE_CONFLICT. `focus` is at most 32 tokens.
`INSERT_SUBTASK`: `mode` must be VERIFY, BRIDGE, or SUPPLEMENT. `task` is at most 48 tokens and is required for BRIDGE and SUPPLEMENT.
`KEEP`, `RETRY_CURRENT`, and `BACKTRACK_PARENT` use an empty payload.
[JSON examples omitted.]
OBSERVATION: {`public_observation_json`}

Training follow-up after KEEP:
Next gate. Apply the same decision rules to this new observation.
OBSERVATION: {`next_gate_observation_json`}

Figure 4: Excerpts from the Runtime Editor prompt. Observations are populated dynamically with public execution information and action constraints. The final block shows the training-time follow-up after KEEP.

LLM-Based Error Attribution Prompt (Who&When)

System_prompt:
You are a precise failure-attribution evaluator. Follow the requested output format exactly and do not add headings, markdown fences, or extra lines.

User_prompt:
Analyze the failed multi-agent trajectory below.
Problem: {question}
Reference correct answer: {reference_answer}
Valid Step Number range: 0 to {last_step}, inclusive.
Identify the single agent directly responsible for the wrong solution and the first zero-based step where that agent made the decisive mistake. Agent Name must exactly match the Agent field shown at the selected step. Give a concise one-line reason.
Return exactly these three lines:
`Agent Name: <name>`
`Step Number: <integer>`
`Reason for Mistake: <reason>`
Conversation:
{complete_trajectory}

Additional user-prompt suffix (MAS-Gate only):
Additional MAS-Gate attribution information:
MAS-Gate is a computation-based failure-attribution method that scores and ranks trajectory steps using their utility and communication influence. Its ranking provides useful information for identifying the decisive error. The following steps received the highest scores and are therefore the most suspicious candidates.
MAS-Gate candidate ranking:
Rank 1: Step {step_1}
...
Rank {effective_k}: Step {step_effective_k}
Use the MAS-Gate ranking as an important reference when identifying the single agent and the first decisive-error step.

Figure 5: LLM prompts for step-level error attribution on Who&When. Direct LLM uses the system and user prompts. MAS-Gate additionally appends the ranked-step guidance after the complete trajectory. Braced fields are replaced with sample-specific content, and the ranking contains up to k entries.

LLM-Based Error Attribution Prompt (AEGIS)

System_prompt:
You are a precise failure-attribution evaluator. Follow the requested two-line output format exactly and do not add headings or markdown.

User_prompt:
Analyze the failed multi-agent trajectory below.
Problem: {question}
Reference correct answer: {reference_answer}
Observed final output: {observed_final_output}
Valid selectable agents: {valid_agent_names}.
User and System turns are context only and cannot be selected. Choose exactly one valid agent. Agent Name must exactly match one name in the valid selectable agents list.
Return exactly these two lines:
`Agent Name: <name>`
`Reason for Mistake: <one-line reason>`
Conversation:
{complete_trajectory}

Additional user-prompt suffix (MAS-Gate only):
MAS-Gate identified the following Top-{effective_k} most suspicious agents, ranked by estimated error risk from highest to lowest. Rank 1 is the most suspicious:
Rank 1: {agent_1}
...
Rank {effective_k}: {agent_effective_k}
Use this ranking together with the complete trajectory and the reference correct answer to identify the agent responsible for the failure.

Figure 6: LLM prompts for agent-level error attribution on AEGIS. Direct LLM uses the system and user prompts. MAS-Gate additionally appends agents ranked by their maximum eligible-step risk after the complete trajectory. Braced fields are replaced with sample-specific content, and the ranking contains up to k entries.

LLM-as-a-judge Prompts (VitaBench)

System Prompt: System Information {env_info} User Complete Instruction {user_instruction}
Background - This is a conversation scenario between a user and an assistant, where the assistant can call tools to retrieve information and complete operations. Tool return results will start with "tool"
- You need to evaluate whether the user instruction has been completed. The user's complete instruction has been broken down into several scoring points (rubrics), and you only need to judge whether each scoring point is satisfied
- Due to the large number of conversation turns, we use sliding window evaluation, where each window shows 10 conversation turns with 2 overlapping turns between windows, but rubric status will be preserved across windows
- You are evaluating window {window_idx} (out of {total_windows} windows total)
- <window_content> contains the conversation content for the current window
- <current_rubrics> contains the current status of all scoring points (true means satisfied, false means not satisfied, all scoring points have an initial status of false)

Task - Update the scoring point rubric status based on the conversation content in the current window
- You can update the status from false to true, if and only if the assistant completed the goal in this window
- You can also update true back to false, if and only if the assistant overturned a previous correct conclusion in this window (but note: if the user's needs themselves changed, such as actively canceling after placing an order, the original evaluation of the ordering requirement should not be overturned)
- You can refer to the "User Complete Instruction" to understand the progress of the current conversation window and avoid unnecessary modifications

Important Notes
- Important: All evaluations are based on whether the assistant's responses and tool call requests complete the goals in the rubrics. User expressions in the conversation are only considered as guidance for the assistant and do not directly affect evaluation standards. Everything is based on the rubric fields!
- Important: Query tool return results are only visible to the assistant and do not represent content recommended by the assistant to users, so they do not directly affect evaluation results either. Everything must be based on the assistant's responses to users after obtaining information! Also note that the Assistant cannot fabricate Tool return results!
- Important: For order-related rubrics (involving order details that must generate orders), you must confirm whether the assistant actually completed the ordering operation. The assistant may mistakenly believe they completed the ordering operation when in fact the tool call failed; or situations where the user states they "can place the order themselves," etc., should all be considered as not meeting the requirements
- For rubrics involving order details such as product quantity or delivery time, the original rubric requirements must be strictly met (no deviation in product quantities, delivery must not be later than the expected time). User compromise behavior does not affect evaluation results (for example, when a user states "fewer items is okay", "I have no objections to the order content" or "later delivery is fine" etc.). These situations should still be considered as not meeting the requirements
- For rubrics involving text content matching of addresses or order notes, apply the functional equivalence principle: as long as the actual content can achieve the same function (such as roughly locating the delivery location or conveying the customer's main needs), it is considered to meet the requirements even if the expression is not completely consistent or lacks some details
- If the current window does not involve a certain rubric, keep its original status unchanged; if the current window involves a rubric but cannot fully determine it, key information can be recorded in the justification for future judgment
- In the justification, record key information related to the current rubric and its corresponding round [x] in an appending manner. If status modifications occur, record the reason using concise language. If the status is not modified, restate the previous reason

Format Requirements
- Your response should be a JSON object containing the following fields:
  - rubric_idx: Unique identifier for the rubric
  - rubric: Restatement of the rubric
  - justification: Explanation of status changes
  - meetExpectation: Updated status (true or false)

User Prompt: <window_content> {window_content} </window_content> <current_rubrics> {current_rubrics_str} </current_rubrics>

Figure 7: LLM-as-a-judge prompts used in VitaBench.