HARNESS OPTIMIZATION FOR MULTI-AGENT SYSTEMS ACROSS NODE, EDGE, AND GRAPH

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems (MAS) show strong potential for complex task solving, but realizing this potential requires effective agent harnesses, which impose infrastructural constraints on execution and collaboration. Existing approaches rely on either manual design, which is labor-intensive and task-dependent, or automatic generation, which can overlook inter-agent and group-level failures, while emphasizing the individual agent harness. Effective MAS harnesses must therefore cover individual execution (*i.e.,* node), inter-agent collaboration (*i.e.,* edge), and group coordination (*i.e.,* graph). However, directly extending the automatic optimization loop of failure diagnosis and constraint generation across these levels introduces two challenges: failure diagnosis duplication, where one underlying failure is repeatedly diagnosed through its propagated symptoms, and constraint candidate explosion, where numerous proposed repairs incur excessive execution costs. We propose MASH, a framework for efficient MAS harness optimization across all three levels. MASH performs structured failure diagnosis by tracing observed symptoms to shared gaps between the information a decision requires and receives, reducing redundant diagnoses and repeated repairs. It then generates coordinated cross-level constraints grounded in each diagnosed failure and applies progressively more expensive quality checks to filter weak candidates before they enter the next optimization round. Experiments across four benchmarks and four MAS topologies demonstrate that MASH consistently outperforms the evaluated baselines, supporting the effectiveness of harness optimization tailored to multi-agent execution. The codes are available at here.

1 INTRODUCTION

Multi-agent systems (MAS) achieve strong task-solving capability by dividing work across specialized agents and executing in parallel, showcasing remarkable performance on tasks such as programming (Hong et al., 2024; Qian et al., 2024), mathematical reasoning (Du et al., 2024; Chen et al., 2024), and deep research (Hadfield et al., 2025; Câmara et al., 2026). However, their performance depends not only on the capabilities of individual models, but also on the ***agent harness***, *i.e.,* the *infrastructural constraints* governing tool use, information exchange, and execution control (Wu et al., 2023). Inadequate constraints can prevent otherwise capable agents from working effectively together, leading to failures in execution, collaboration, and verification (Cemri et al., 2025; Ning et al., 2026). Therefore, it is essential to build an effective MAS harness, unleashing the full potential of MAS and facilitating real-world applications.

Recent research typically achieves harness (*i.e.,* infrastructural constraints) construction through an iterative optimization loop (Lee et al., 2026b; Lin et al., 2026; Chen et al., 2026b; Zhang et al., 2026b) as shown in Figure 1(a). Specifically, given an agent task (*e.g.,* deep research), the optimization loop typically contains three steps: 1) *task execution*, which runs agents under each proposed infrastructural constraint and collects trajectories[1]. Based on execution results, useful constraints are incorporated into the harness. 2) *failure diagnosis*, which examines the collected trajectories to diagnose infrastructural failures (*e.g.,* evidence lost during agent handoffs) and determine whether the proposed constraints address them; 3) *constraint generation*, which proposes new constraints based

[1]In the initial optimization round, agents execute under the original harness.

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

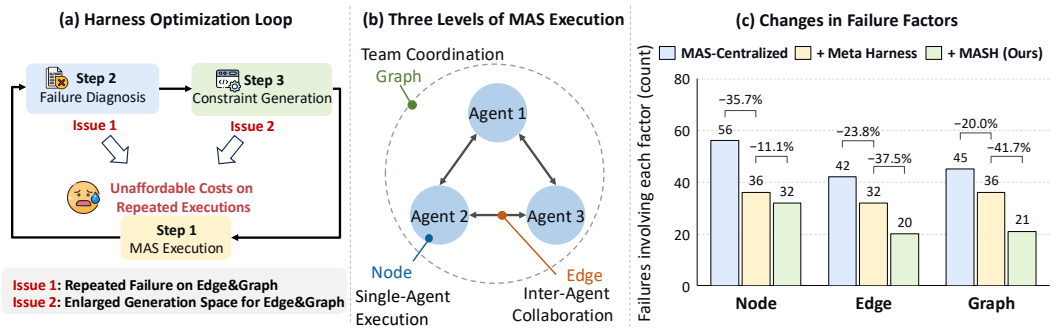


Figure 1: Motivation for MAS harness optimization. (a) Extending optimization to MAS introduces redundant diagnosis and more constraint candidates, increasing execution costs. (b) MAS execution spans node, edge, and graph levels with inter-agent information exchange. (c) Existing harness optimization is less effective at addressing edge- and graph-level failures than node-level failures.

on the diagnosed failures. With the generated constraints, it then goes back to the first step (*i.e.,* task execution) for next-round optimization.

Despite their effectiveness, existing automatic harness-construction methods fail to generalize well to MAS, yielding only partial harnesses. Specifically, existing methods primarily diagnose failures and generate constraints around an individual agent's interaction with its environment. Yet, MAS execution involves three levels, including single-agent execution (*i.e.,* node), inter-agent collaboration (*i.e.,* edge), and agent group coordination (*i.e.,* graph). Directly applying existing methods can over-emphasize the constraints on single-agent execution, thus leaving failures in edge and graph levels unfixed (see Figure 1 for empirical evidence). Therefore, it is essential to build infrastructural constraints for MAS explicitly across the node, edge, and graph levels.

A natural solution is to explicitly diagnose failures across the three levels and then propose level-specific constraints. However, in MAS, interactions among agents allow a node-level failure to propagate to the edge and graph levels. Diagnosing these failures can produce repeated diagnoses and similar constraint candidates that target the same underlying issue. The key is therefore to retain a small set of high-quality candidates while eliminating redundant or ineffective ones. To achieve this, we posit two principles for failure diagnosis and constraint generation steps, respectively: 1) **Failure deduplication**, which aims to reduce repeated diagnoses that may generate similar constraints. Repeated diagnoses arising from failure propagation can yield similar constraint candidates, each requiring a separate MAS execution to collect trajectories for the next round. This wastes computation on the same underlying issue. It is therefore important to merge diagnoses that share a root cause, retaining one diagnosis per distinct cause. 2) **Constraint candidate filtering**, which aims to avoid expensive MAS executions on low-quality candidates. Generating constraints across three levels can produce a large number of constraint candidates, each requiring a separate MAS execution for next-round diagnosis. Passing all candidates into subsequent rounds therefore causes overwhelming computational burdens. To address this issue, the key is to quickly filter out low-quality candidates before candidates enter the next round, reserving execution budget for promising constraints.

Following these principles, we propose **MASH**, an efficient MAS Harness generation framework across node, edge, and graph. At the *failure diagnosis step*, we introduce a structured failure diagnosis strategy, which aims to trace the root cause of failures and reduce repeated diagnosis. It organizes each diagnosis as a structured tuple indicating what required information is missing and where the information gap first arises during execution (detailed descriptions are in Section 4.2). By grouping structured diagnoses that concern the same required information, MASH reduces repeated diagnoses and avoids generating similar constraints. For *the constraint generation step*, MASH first generates constraints explicitly for node, edge, and graph levels to cover the constraint space of MAS. To prevent low-quality candidates from entering the next-round execution, we introduce a cascaded filtering strategy consisting of checks on syntax (*i.e.,* whether the generated constraint is executable), effectiveness (*i.e.,* whether the constraint is useful in the specific case), and robustness (*i.e.,* whether the constraint generalizes across diverse tasks), respectively. The remaining candidates that pass the checks will then be executed and diagnosed for the next round. We evaluate MASH on

four datasets (*i.e.,* BrowseComp-Plus, WorkBench, Plancraft, Finance) across multiple workflows and LLM backbones, demonstrating its effectiveness across individual-agent execution, inter-agent interactions, and team coordination, with an average reduction in node, edge, and graph-level failures of 7.7%, 25.5%, and 29.1%, respectively. Extensive analyses of constraint quality, optimization efficiency, and generalization further validate the effectiveness, efficiency, and generalizability of our proposed method.

The main contributions of this work are summarized as follows:

- We motivate harness optimization across node, edge, and graph levels in MAS and formulate two guiding principles, failure deduplication and constraint candidate filtering, to reduce redundant proposals and unnecessary MAS executions.
- We propose MASH, an efficient MAS harness optimization framework that combines structured failure diagnosis with cross-level constraint generation and fast filtering to retain high-quality constraints while reducing optimization costs.
- We conduct extensive experiments on Finance-Agent, BrowseComp-Plus, WorkBench, and Plancraft across multiple workflows and backbone LLMs, demonstrating the effectiveness, efficiency, and robustness of MASH.

2 Related Work

2.1 LLM-based Multi-Agent Systems

LLM-based MAS improve task solving through specialized roles, task decomposition, and inter-agent communication. Frameworks such as MetaGPT and ChatDev organize agents through structured collaboration protocols (Hong et al., 2024; Qian et al., 2024), while subsequent work explores communication topologies and coordination strategies (Zhou et al., 2025; Li et al., 2026). Their benefits depend on task structure, model capability, and coordination overhead (Kim et al., 2026). However, distributing execution also introduces opportunities for information loss, inter-agent misalignment, and incomplete verification (Hadfield et al., 2025; Cemri et al., 2025). Empirical failure analyses identify system design and coordination deficiencies as important sources of unsuccessful execution, beyond limitations of individual models (Zhang et al., 2025e; 2026a; Kong et al., 2026), while TraceElephant emphasizes the importance of complete execution context for diagnosis (Chen et al., 2026a). These observations motivate examining the infrastructure governing agent behavior and interaction alongside the underlying models. Our work constructs a harness for MAS: infrastructural constraints that regulate individual execution, inter-agent information exchange, and group coordination to address failures across these interaction levels.

2.2 Agent Harness Optimization

Agent harnesses govern tool access, context management, and execution control (Ning et al., 2026; Lee et al., 2026b; Ursekar et al., 2026a). Beyond manual design, recent methods use execution feedback to optimize prompts and context (Agrawal et al., 2026; Zhang et al., 2025d; Kang et al., 2026), memory and tools (Zhang et al., 2025b; 2026c), and broader executable harness components (Lee et al., 2026b; Lin et al., 2026; Xu et al., 2026b). Further work improves candidate generation and verification (Chen et al., 2026b; Zhang et al., 2026b; Xu et al., 2026a; Wu et al., 2026), or adapts harnesses across successive tasks and interactions (He et al., 2026; Karten et al., 2026; Wei et al., 2026). We provide a detailed discussion in Appendix D.1. In MAS, harness optimization needs to additionally address failures in inter-agent collaboration and group coordination. Failure propagation across these levels can produce repeated diagnoses and similar constraint candidates, increasing evaluation costs.

2.3 Multi-Agent Workflow Optimization and Evolution

Prior work improves multi-agent systems through workflow search and architecture selection (Zhang et al., 2025c;a; Wang et al., 2025), coordinated prompt optimization (Hong et al., 2026; Wang et al., 2026b), and the evolution of collaboration mechanisms and team organization (Lee et al., 2026a; Hao et al., 2026). Data Canvas further uses provenance to trace errors and selectively replay affected

workflow components (Yi et al., 2026). Building on these advances, MASH focuses on efficient harness optimization under fixed agents, communication topology, and execution budgets. It links failures across levels to shared root causes and combines coordinated constraint generation with fast filtering, reducing repeated diagnosis and unnecessary candidate evaluation.

3 Problem Formulation

MAS harness. An MAS $\mathcal{M}$ executes tasks under a harness H, a set of infrastructural constraints governing its execution. MAS involve not only interactions between individual agents and their environments, but also information exchange among agents and coordination across the group. We therefore define an MAS harness across three levels:

$$H = (C^{\mathrm{n}}, C^{\mathrm{e}}, C^{\mathrm{g}}),$$

where node constraints C^{n} govern individual agents' execution, edge constraints C^{e} govern inter-agent information exchange, and graph constraints C^{g} govern group coordination.

Harness optimization objective. Our objective is to find a harness that maximizes expected task performance:

$$H^{\star} \in \arg\max_{H \in \mathcal{H}} \mathbb{E}_{x \sim \mathcal{D}} \left[s\big(\mathrm{Exec}(\mathcal{M}[H], x)\big) \right],$$

where $\mathcal{D}$ is the task distribution, Exec returns an execution trajectory, s scores task performance in $[0, 1]$, and $\mathcal{H}$ is the space of valid harnesses. During optimization, we hold the backbone models, agents, base roles, communication topology, tool permissions, and per-task budgets fixed.

Harness optimization loop. This objective is typically pursued through a 3-step optimization loop:

1. *Task execution.* The MAS executes tasks sampled from $\mathcal{D}$ under the current harness H_t, producing trajectories $\mathcal{T}_t = \mathrm{Exec}(H_t; \mathcal{D})$.
2. *Failure diagnosis.* The collected trajectories are examined to diagnose infrastructural failures, $\mathcal{F}_t = \mathrm{Diagnose}(\mathcal{T}_t)$.
3. *Constraint generation.* The diagnosed failures guide candidate constraint changes, $\mathcal{C}_t = \mathrm{Generate}(H_t, \mathcal{F}_t)$. Each candidate may revise constraints at one or more of the node, edge, and graph levels.

Each candidate $c \in \mathcal{C}_t$ is applied separately to H_t, yielding a candidate harness $H_t \oplus c$ for task execution and failure diagnosis in the next round, where $\oplus$ denotes applying the proposed constraint changes. Based on execution results, ineffective changes are discarded and useful ones are incorporated into the harness, which supports further constraint generation.

4 Our Method (MASH)

4.1 Overview

MASH starts from the original harness H_0 and repeats the optimization loop for T rounds. To reduce *failure diagnosis duplication*, structured failure diagnosis relates symptoms across levels to shared information gaps and checks them against prior changes (§4.2). To achieve *effective constraint generation* to avoid computationally expensive task execution on low-quality constraints, constraint generation and filtering jointly determine which changes enter the next round (§4.3): each candidate addresses one shared problem through constraints at the required levels, and progressively more expensive checks discard weak candidates. At most one candidate replaces the current harness per round. The final harness H_T is frozen before test evaluation.

4.2 Structured Failure Diagnosis

Failure diagnosis in MAS must cover node, edge, and graph levels, yet failures at different levels may originate from the same node-level error propagated through agent interactions. Diagnosing them

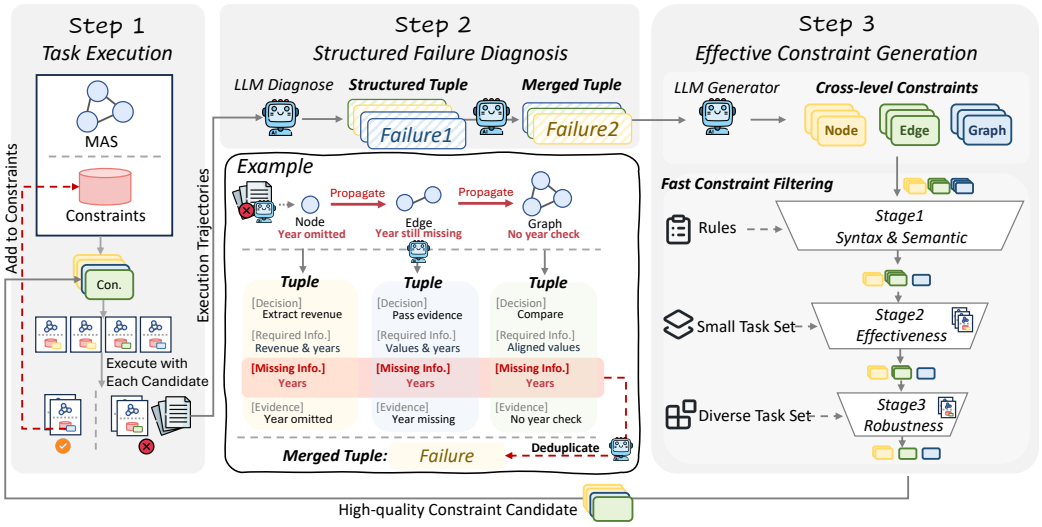


Figure 2: Overview of MASH. (1) Task execution applies each candidate constraint separately to the current harness and incorporates useful constraints based on execution results. (2) Structured failure diagnosis organizes failures into tuples and merges diagnoses supported by a shared root cause. (3) Effective constraint generation first generates node-, edge-, and graph-level constraints, then filters candidates through syntax and semantics, effectiveness, and robustness checks before the next round.

independently can yield similar constraints and redundant MAS executions. The goal is therefore to trace failures to their root causes and consolidate diagnoses of the same underlying issue.

MASH implements $\mathrm{Diagnose}(\mathcal{T}_t)$ through an LLM that summarizes failures in the execution trajectories as structured tuples, traces their causes, and consolidates related diagnoses within the same process. Each tuple records what information a failed decision requires, what it receives, and where the required information first becomes missing. The resulting diagnosis set is

$$\mathcal{F}_t = \mathrm{Diagnose}(\mathcal{T}_t) = \{(D_i, R_i, I_i, M_i, E_i)\}_i,$$

where i indexes the consolidated diagnoses, D_i is the failed *decision*, R_i the *required information*, I_i the *input information* actually received, M_i the *missing information*, and E_i the supporting trajectory *evidence*. These fields guide the LLM to trace missing information backward through agent outputs and inter-agent exchanges, summarizing failures with a shared root cause into one diagnosis.

For example, an agent may omit the reporting year of a revenue figure, which is then passed to other agents and used in a comparison across different years. Tracing these failures to the original omission yields a shared diagnosis, avoiding similar constraint proposals for the same cause. To avoid combining distinct failures, the LLM consolidates diagnoses only when trajectory evidence supports a common root cause, not merely the same missing information.

4.3 Constraint Generation and Filtering

Generating constraints across three levels expands the generation space of infrastructural constraints. MASH limits redundant proposals by organizing each candidate around one diagnosed problem, then filters candidates before spending further MAS executions on them.

4.3.1 Cross-Level Constraint Generation

Given the diagnosed failures $\mathcal{F}_t$ and current constraints H_t, MASH implements constraint generation using an LLM:

$$\mathcal{C}_t = \mathrm{Generate}(H_t, \mathcal{F}_t).$$

Each generated constraint is explicitly mapped to the node, edge, or graph level according to its target: individual-agent execution, inter-agent information exchange, or group coordination, respectively. This mapping organizes the proposed changes across the three levels, allowing constraints addressing a shared failure cause to span multiple levels. Implementation details are provided in Appendix A.2.

4.3.2 FAST CONSTRAINT FILTERING

Generating constraints across node, edge, and graph expands the candidate space. Passing all candidates into subsequent optimization rounds would require many MAS executions, slowing optimization. We therefore assess candidate quality before further iteration. A useful constraint should satisfy three requirements: *syntactic and semantic validity*, *effectiveness* in addressing diagnosed failures, and *robustness* across tasks. MASH introduces a three-stage filtering process ordered by the cost of verifying these requirements. Candidates proceed to the next stage only after passing the preceding check.

Stage 1: syntax and semantics check. MASH first applies rule-based checks to each candidate's implementation and proposed changes. Syntax checks detect invalid implementations, while semantic checks examine compliance with execution rules and consistency with the intended changes. Candidates containing forbidden operations, missing changes, or changes outside the proposal are discarded. This stage retains valid and consistent candidates without requiring any MAS task execution.

Stage 2: effectiveness check. Surviving candidates are tested on a small subset of diagnosis tasks, denoted by $\mathcal{D}_{\text{small}}$. Each candidate $c \in \mathcal{C}_t$ is applied separately to the current harness, yielding $H' = H_t \oplus c$. The candidate harness H' and the current harness H_t execute the same tasks. For a task set $\mathcal{X}$, their average performance difference is

$$\Delta_{\mathcal{X}}(H', H_t) = \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \big[s_x(H') - s_x(H_t) \big],$$

where $s_x(H)$ is the task score under harness H. Candidates are retained if $\Delta_{\mathcal{D}_{\text{small}}}(H', H_t) \geq \theta_{\text{eff}}$, where θ_{eff} is a predefined effectiveness threshold. This small-sample check removes candidates with insufficient gains before more expensive testing.

Stage 3: robustness check. Candidates passing the effectiveness check are evaluated on a larger, diverse validation set $\mathcal{D}_{\text{val}}$, separate from the diagnosis tasks. This stage tests whether their benefits extend beyond the failures that motivated them. Candidates are retained if $\Delta_{\mathcal{D}_{\text{val}}}(H', H_t) \geq \theta_{\text{gen}}$, where θ_{gen} is a predefined robustness threshold. Candidates whose improvements fail to transfer sufficiently to these tasks are discarded.

Only candidates passing all three stages enter the next optimization round for task execution and failure diagnosis. By rejecting invalid candidates without MAS execution and ineffective candidates using only a small task subset, MASH reserves broader evaluation and further iteration for high-quality constraints.

4.4 OPTIMIZATION LOOP OF MASH

Each round takes the current harness H_t and candidate constraints retained from the preceding round as input. During 1) *task execution*, each candidate c is applied separately to H_t, and MAS execution under $H_t \oplus c$ produces trajectories $\mathcal{T}_t$. Based on execution results, MASH incorporates the effective candidate into the harness to obtain H_{t+1}, retaining H_t if no candidate is effective. The initial round executes under the original harness H_0. During 2) *failure diagnosis*, $\mathcal{F}_t = \text{Diagnose}(\mathcal{T}_t)$ organizes failures into structured tuples and consolidates diagnoses supported by shared root causes. During 3) *constraint generation*, $\mathcal{C}_t = \text{Generate}(H_{t+1}, \mathcal{F}_t)$ proposes candidate constraints across node, edge, and graph. The three-stage filtering process then removes low-quality candidates. Each round thus outputs the updated harness H_{t+1} and surviving candidates from $\mathcal{C}_t$ for the next round's task execution. After T rounds, the final harness is frozen for test evaluation. Algorithm 1 in the appendix summarizes the procedure.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETTINGS

Benchmarks and baselines. We evaluate MASH on Finance-Agent (Bigeard et al., 2025), BrowseComp-Plus (Chen et al., 2026c), Plancraft (Dagan et al., 2025), and WorkBench (Styles

Table 1: Mean test scores ($\times 100$; higher is better) over three repetitions under the centralized workflow. Baselines are grouped by their primary optimization scope. Best scores are bold; best baseline scores are underlined. Improvement is MASH minus the best baseline in absolute score points.

Method	Finance-Agent	BrowseComp-Plus	Plancraft	WorkBench
Static reference				
MAS-Centralized (Kim et al., 2026)	39.97	20.00	53.00	35.00
Automatic harness/context optimization				
GEPA (Agrawal et al., 2026)	39.32	21.00	51.67	34.67
RHI (Lee et al., 2026a)	43.02	23.00	54.33	38.00
ACE (Zhang et al., 2025d)	38.23	20.33	51.00	33.67
AHE (Lin et al., 2026)	46.83	24.00	55.33	40.00
Life-Harness (Xu et al., 2026b)	43.49	21.67	54.00	42.00
Meta-Harness (Lee et al., 2026b)	47.24	25.00	56.33	42.33
HarnessX (Chen et al., 2026b)	<u>48.76</u>	<u>25.33</u>	56.00	<u>43.33</u>
MAS/workflow optimization				
Meta-Team (Hao et al., 2026)	41.86	21.67	55.00	40.67
AFlow (Zhang et al., 2025c)	42.85	22.33	<u>57.00</u>	39.00
Ours				
MASH	**52.88**	**29.00**	**61.67**	**48.00**
Improvement	+4.12	+3.67	+4.67	+4.67

et al., 2024). To evaluate our method, we compare MASH with three groups of baselines. The *static reference* is the initial MAS without additional harness optimization. *Automatic harness and context optimization* methods include GEPA (Agrawal et al., 2026), RHI (Lee et al., 2026a), ACE (Zhang et al., 2025d), AHE (Lin et al., 2026), Life-Harness (Xu et al., 2026b), Meta-Harness (Lee et al., 2026b), and HarnessX (Chen et al., 2026b). *MAS and workflow optimization* methods include Meta-Team (Hao et al., 2026) and AFlow (Zhang et al., 2025c). Detailed descriptions and adaptation protocols for all baselines are provided in Appendix B.4.

Implementation details. The main comparison uses a centralized workflow (Kim et al., 2026), and all methods share the same agent roles, topology, tools, and LLM backbone (Gemini 2.5 Flash-Lite with temperature = 0). MASH uses GPT-5.4 for offline *structured failure diagnosis* and *cross-level constraint generation*, with separate prompts for the two functions. All methods share a development task-attempt cap within each benchmark (4,000 per fold on Finance-Agent), while retaining method-specific schedules and stopping rules. Optimizer requests and tokens are accounted for separately; see Appendix B.4 for details. MASH runs for at most $T = 20$ optimization loops, with at most three initially generated constraints per optimization loop. Data splits are detailed in Appendix B.3. After optimization, the selected harness is frozen and evaluated on $\mathcal{D}_{\text{test}}$ over three test repetitions; we report the mean test score across repetitions.

5.2 Main Results

Table 1 shows that MASH achieves the highest mean test score on all four benchmarks across the static, automatic harness/context, and MAS/workflow comparison groups. The strongest baseline varies across benchmarks: HarnessX provides the highest baseline score on Finance-Agent, BrowseComp-Plus, and WorkBench, whereas AFlow provides the highest baseline score on Plancraft. The gains over the initial MAS and the evaluated optimization baselines support optimizing infrastructural constraints while keeping the underlying MAS configuration and per-task execution limits fixed.

Table 2: Failure counts: Meta-Harness → MASH. Superscripts show reductions; the last row averages them.

Benchmark	Node	Edge	Graph
Finance-Agent	$36 \rightarrow 32^{11.1\%}$	$32 \rightarrow 20^{37.5\%}$	$36 \rightarrow 21^{41.7\%}$
BrowseComp-Plus	$113 \rightarrow 108^{4.4\%}$	$78 \rightarrow 56^{28.2\%}$	$74 \rightarrow 55^{25.7\%}$
Plancraft	$85 \rightarrow 77^{9.4\%}$	$35 \rightarrow 32^{8.6\%}$	$65 \rightarrow 50^{23.1\%}$
WorkBench	$87 \rightarrow 82^{5.7\%}$	$54 \rightarrow 39^{27.8\%}$	$89 \rightarrow 66^{25.8\%}$
Failure reduction	**7.7%**	**25.5%**	**29.1%**

Table 3: Ablation study of MASH on Finance-Agent.

	Structured diagnosis	Cross-level generation	Acc(%)
Variants	×	×	48.60
	×	✓	50.57
	✓	×	51.44
MASH	✓	✓	**52.88**



Figure 3: Analysis of MASH. (a) MASH achieves higher constraint acceptance and performance improvement rates. (b) MASH enables more effective executions with less wasted computation. (c) Structured failure diagnosis reduces redundant diagnoses. (d) A case study showing how three linked failures across node, edge, and graph levels are consolidated into a shared root cause to guide constraint generation.

5.3 In-depth Analysis of MASH

Benefits of MASH across node, edge, and graph. To examine the value of constraints beyond individual agents, we compare failure patterns and vary the levels available for optimization. Table 2 shows that MASH reduces edge- and graph-level failures more substantially than node-level failures relative to Meta-Harness. Moreover, allowing all three levels to remain editable consistently outperforms node-only optimization, whereas freezing node constraints and optimizing only the edge and graph levels is not uniformly beneficial (Appendix C.4).

Ablation study. To assess the two components, we replace structured failure diagnosis with free-form diagnosis and restrict cross-level constraint generation to single-level candidates. Table 3 shows that both components improve performance, with structured diagnosis contributing larger gains. This may be because propagated failures can lead to ineffective constraints, which can in turn negatively affect subsequent optimization. A more detailed analysis of the effects of the hyperparameters is presented in Appendix C.7.

Reduced redundancy in failure diagnosis. Figure 3(c) shows that structured failure diagnosis reduces failure overlap from 38.62% to 22.73% compared with free-form diagnosis. Figure 3(d) illustrates how multiple failure symptoms are traced to a shared coordination gap to guide constraint generation. The full procedure is provided in Appendix C.6.

Constraint quality analysis. To examine the quality of generated constraints, we compare their improvement rates among candidates completing paired evaluation. Figure 3(a) shows that both structured diagnosis and cross-level generation lead to higher-quality constraints, thereby improving MAS task performance (*i.e.,* results are consistent with ablation study). Moreover, most accepted constraints involve inter-agent collaboration or group coordination, highlighting their practical use beyond node-level execution (Appendix C.5).

Optimization efficiency. We analyze optimization efficiency through: 1) *LLM calls for candidate evaluation.* Syntax-and-semantics and effectiveness filtering reduce unnecessary execution: removing them increases LLM calls by 9.7% and 12.3%, respectively. Robustness filtering improves final performance at a modest cost, with its removal saving only 1.7% of calls (see Appendix B.1.3 for detailed results). 2) *Rate of harness improvement.* Compared with diagnosing failures and generating

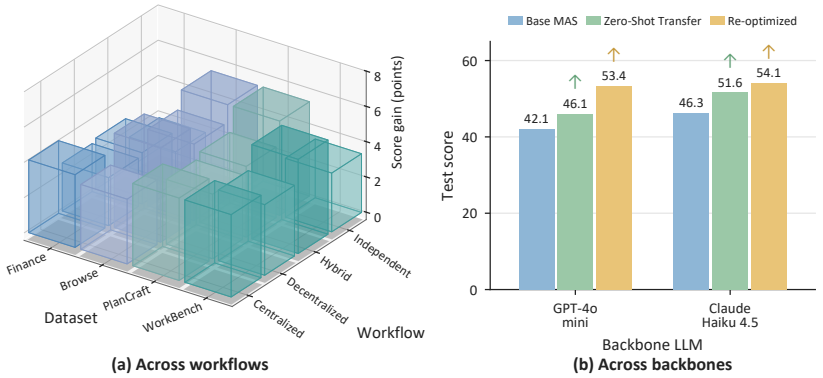


Figure 4: Generalization of MASH across workflows and LLM backbones. (a) Test-score gains over the best reported baseline across four benchmarks and four fixed workflows. (b) Zero-shot transfer of MASH constraints across LLM backbones, followed by target-side re-optimization.

constraints separately at the node, edge, and graph levels, MASH requires fewer optimizer tokens per loop and improves performance more rapidly, finding effective harnesses sooner and achieving better final performance (see Appendix C.8 for detailed results).

5.4 Generalization Analysis

Generalization across workflows. We optimize harnesses independently under four fixed workflows: centralized, decentralized, hybrid, and independent (Appendix B.2.1). Figure 4(a) shows gains over the best reported baseline in all 16 benchmark–workflow settings, demonstrating that MASH is effective across communication topologies, including workflows without intermediate communication.

Generalization across backbones. We transfer harnesses optimized on Gemini 2.5 Flash-Lite to GPT-4o mini and Claude Haiku 4.5 (Appendix B.2.2). Figure 4(b) shows that frozen transfer improves performance on both backbones, while target-side optimization yields further gains, supporting both the transferability and adaptability of the learned constraints.

6 Conclusion

We presented MASH, a harness optimization framework for MAS across node, edge, and graph levels. MASH traces failures to shared information gaps through structured failure diagnosis, generates cross-level constraints according to their execution targets, and filters candidates through progressively more expensive checks to reduce unnecessary MAS executions. Experiments across four benchmarks, multiple workflows, and LLM backbones demonstrate its effectiveness. Future work includes jointly optimizing harnesses and communication topology, reducing the evaluation cost of surviving candidates, automatically assigning constraint levels for arbitrary MAS, and consolidating accumulated constraints when later revisions supersede earlier ones.

AI Use Statement

In this work, we have not used generative AI tools for generating synthetic datasets, formulating mathematical claims, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, assisting with translation, cleaning and reformatting datasets, supporting qualitative and thematic data analysis, interpreting results. Providing critical ingredients for proving mathematical claims and assisting in the writing of proofs are not applicable to this work. Additionally, we used generative AI tools for creating or editing software code, writing refinement to improve readability, and identifying relevant literature. We have reviewed all AI-assisted work. In particular, all literature identified or listed with the assistance of generative AI tools was independently checked by the authors for accuracy and relevance. LLM-generated code was reviewed, tested, and modified

by the authors to verify its correctness and ensure that it implemented the intended functionality. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2507.19457.

Antoine Bigeard, Langston Nashold, Rayan Krishnan, and Shirley Wu. Finance agent benchmark: Benchmarking llms on real-world financial research tasks, 2025. URL https://arxiv.org/abs/2508.00828.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *arXiv preprint arXiv:2503.13657*, 2025.

Justin Chen, Swarnadeep Saha, and Mohit Bansal. ReConcile: Round-table conference improves reasoning via consensus among diverse LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7066–7085, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.381. URL https://aclanthology.org/2024.acl-long.381/.

Mengzhuo Chen, Junjie Wang, Fangwen Mu, Yawen Wang, Zhe Liu, Huanxiang Feng, and Qing Wang. Seeing the whole elephant: A benchmark for failure attribution in LLM-based multi-agent systems. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19888–19905, San Diego, California, United States, July 2026a. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.912. URL https://aclanthology.org/2026.acl-long.912/.

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, Yuan Xie, Heng Qu, Kun Shao, and Jian Luan. HarnessX: A composable, adaptive, and evolvable agent harness foundry. *arXiv preprint arXiv:2606.14249*, 2026b. URL https://arxiv.org/abs/2606.14249.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Sahel Sharifymoghaddam, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Hosna Oyarhoseini, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. BrowseComp-Plus: A fair and disentangled evaluation benchmark for deep search agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22349–22370, 2026c. doi: 10.18653/v1/2026.acl-long.1023.

Arthur Câmara, Vincent Slot, and Jakub Zavrel. Self-optimizing multi-agent systems for deep research, 2026. URL https://arxiv.org/abs/2604.02988.

Gautier Dagan, Frank Keller, and Alex Lascarides. Plancraft: An evaluation dataset for planning with llm agents. In *The 2nd Conference on Language Modeling*, 2025.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Jeremy Hadfield, Barry Zhang, Kenneth Lien, Florian Scholz, Jeremy Fox, and Daniel Ford. How we built our multi-agent research system. Anthropic Engineering, June 2025. URL https://www.anthropic.com/engineering/multi-agent-research-system. Published June 13, 2025. Engineering report; accessed September 22, 2026.

Zhezheng Hao, Tianfu Wang, Huanshuo Dong, Ziyan Liu, Hong Wang, Xiankun Lin, Qiang Lin, Can Wang, Hande Dong, and Jiawei Chen. Evolve as a team: Collaborative self-evolution for LLM-based multi-agent systems. *arXiv preprint arXiv:2605.29790*, 2026. URL https://arxiv.org/abs/2605.29790.

Yufei He, Juncheng Liu, Yue Liu, Yibo Li, Tri Cao, Zhiyuan Hu, Xinxing Xu, and Bryan Hooi. Evotest: Evolutionary test-time learning for self-improving agentic systems. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 104166–104202, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/aa5b64942bfef625403b355214b871e0-Paper-Conference.pdf.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, zili wang, Steven Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/6507b115562bb0a305f1958ccc87355a-Paper-Conference.pdf.

Zhi Hong, Qian Zhang, Jiahang Sun, Zhiwei Shang, Mingze Kong, Xiangyi Wang, Yao Shu, and Zhongxiang Dai. Maspob: Bandit-based prompt optimization for multi-agent systems with graph neural networks, 2026. URL https://arxiv.org/abs/2603.02630.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. Acon: Optimizing context compression for long-horizon llm agents, 2026. URL https://arxiv.org/abs/2510.00615.

Seth Karten, Joel Zhang, Tersoo Upaa Jr, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual harness: Online adaptation for self-improving foundation agents, 2026. URL https://arxiv.org/abs/2605.09998.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, et al. Capable language models can outgrow the benefits of collaboration. *Nature Machine Intelligence*, 8(7):1157–1172, 2026.

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.

Hyunin Lee, Jinglue Xu, Jeffrey Seely, Donghyun Lee, Matei Zaharia, and Yujin Tang. Recursive harness self-improvement. *arXiv preprint arXiv:2607.15524*, 2026a. URL https://arxiv.org/abs/2607.15524.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026b. URL https://arxiv.org/abs/2603.28052.

Shiyuan Li, Yixin Liu, Qingsong Wen, Chengqi Zhang, and Shirui Pan. Assemble your crew: Automatic multi-agent communication topology design via autoregressive graph generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, 2026.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, et al. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026.

Xiaotian Luo, Dizhan Xue, Fengxingyu Wang, Chuanrui Hu, and Yafeng Deng. Harnessbank: Semantic gene-bank search with gated verification for agent-harness self-evolution, 2026. URL https://arxiv.org/abs/2607.13683.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, Lingjie Chen, Yanjun Zhao, Ke Yang, Bingxuan Li, Cheng Qian, Gaotang Li, Xiao Lin, Zhichen Zeng, Ruizhong Qiu, Sirui Chen, Yifan Sun, Xiyuan Yang, Ruida Wang, Rui Pan, Chenyuan Yang, Dylan Zhang, Liri Fang, Zikun Cui, Yang Cao, Pan Chen, Dorothy Sun, Ren Chen, Mahesh Srinivasan, Nipun Mathur, Yinglong Xia, Hong Li, Hong Yan, Pan Lu, Lingming Zhang, Tong Zhang, Hanghang Tong, and Jingrui He. Code as agent harness, 2026. URL https://arxiv.org/abs/2605.18747.

Sungho Park, Wonjoong Kim, Rongyuan Tan, Jue Zhang, Wook-Shin Han, Pengfei Gao, Chanyoung Park, Yongqiang Yao, Rao Fu, Elsie Nallipogu, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Autosaddler: Automatic harness optimization with durable updates from agent execution traces, 2026. URL https://arxiv.org/abs/2608.23041.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. Workbench: A benchmark dataset for agents in a realistic workplace setting. In *Conference on Language Modeling*, 2024.

Varun Ursekar, Apaar Shanker, Veronica Chatrath, Yuan Xue, and Samuel Marc Denton. Vero: A harness for agents to optimize agents, 2026a. URL https://arxiv.org/abs/2602.22480.

Varun Ursekar, Apaar Shanker, Yash Maurya, Shehab Yasser, Vijay S. Kalmath, Veronica Chatrath, and Yuan Xue. Harnessopt-bench: Evaluating llms at harness optimization, 2026b. URL https://arxiv.org/abs/2608.06301.

Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Pradeep Dasigi, and Teng Xiao. Rethinking the evaluation of harness evolution for agents, 2026a. URL https://arxiv.org/abs/2607.12227.

Yingxu Wang, Siwei Liu, Jinyuan Fang, and Zaiqiao Meng. Evoagentx: An automated framework for evolving agentic workflows. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2025.

Zhexuan Wang, Xuebo Liu, Li Wang, Zifei Shan, Yutong Wang, Zhenxi Song, and Min Zhang. Maspo: Joint prompt optimization for llm-based multi-agent systems, 2026b. URL https://arxiv.org/abs/2605.06623.

Tianxin Wei, Zhan Shi, Minhua Lin, Bing He, Zewen Liu, Yisi Sang, Yuanchen Bei, Xuying Ning, Jiaru Zou, Ting-Wei Li, Xiao Lin, Yanjun Zhao, Chi Wang, Benoit Dumoulin, Dakuo Wang, Jingrui He, and Hanqing Lu. Evo-harness: Context-to-harness skill compilation for self-evolving agents, 2026. URL https://arxiv.org/abs/2608.15071.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Siwei Wu, Jincheng Ren, Yizhi Li, Haau-Sing Li, Chengran Yang, Yuxuan Zhang, Weicheng Gu, Jian Yang, Riza Batista-Navarro, Chuanyi Zhang, Xianglong Liu, Ming Zhou, Bryan Dai, and Chenghua Lin. Modularrsi: Modular and generalizable recursive harness self-improvement, 2026. URL https://arxiv.org/abs/2609.14857.

Jinghan Xu, Yikai Zhang, Aili Chen, Weiyuan Li, Jiaqing Liang, and Deqing Yang. Verify smarter, evolve further: Efficient harness evolution through behavior-aware verification, 2026a. URL https://arxiv.org/abs/2608.27311.

Tianshi Xu, Huifeng Wen, and Meng Li. Adapting the interface, not the model: Runtime harness adaptation for deterministic LLM agents. *arXiv preprint arXiv:2605.22166*, 2026b. URL https://arxiv.org/abs/2605.22166.

Zixuan Yi, Yuanming Shao, Shaun Wallace, Zachary G. Ives, and Ryan Marcus. Data Canvas: A Provenance-Guided Harness for Agentic Data Engineering. COLM '26, July 2026. URL https://openreview.net/forum?id=t89YKsfjKC.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 75834–75852. PMLR, 13–19 Jul 2025a. URL https://proceedings.mlr.press/v267/zhang25bi.html.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. Memevolve: Meta-evolution of agent memory systems, 2025b. URL https://arxiv.org/abs/2512.18746.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng YAN. Agentracer: Who is inducing failure in the llm agentic systems? In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 11377–11399, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/134ed7a477770f227f12450ef0cbb8f4-Paper-Conference.pdf.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-Harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026b. URL https://arxiv.org/abs/2606.09498.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations (ICLR)*, 2025c.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025d. URL https://arxiv.org/abs/2510.04618.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *International Conference on Machine Learning (ICML)*, 2025e.

Yuxuan Zhang, Cong Wei, Penghui Du, Bo Li, Huaisong Zhang, Songcheng Cai, Yubo Wang, Dongfu Jiang, Yuyu Zhang, Changqian Yu, Ping Nie, Wenhu Chen, and Kelsey R. Allen. Rewardharness: Learning human preferences for image editing with only 100 demonstrations. In *Conference on Language Modeling*, 2026c.

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Ö. Arık. Multi-agent design: Optimizing agents with better prompts and topologies. In *arXiv preprint arXiv:2502.02533*, 2025.

A METHOD DETAILS

A.1 MASH OPTIMIZATION PROCEDURE

The procedure of MASH is illustrated in Algorithm 1.

Algorithm 1 Optimization Loop of MASH

Input: MAS $\mathcal{M}$, initial harness H_0, diagnosis tasks $\mathcal{D}_{\text{dis}}$, screening subset $\mathcal{D}_{\text{small}}$, validation tasks $\mathcal{D}_{\text{val}}$, thresholds $\theta_{\text{eff}}, \theta_{\text{gen}}$, optimization-loop budget T
Output: Optimized harness H_T

Initialize an empty candidate list
for $t = 0, \ldots, T-1$ **do**
Step 1: Task execution
    $H_{t+1} \leftarrow H_t$
    if the candidate list is empty **then**
        $\mathcal{T}_t \leftarrow \text{Exec}(H_t; \mathcal{D}_{\text{dis}})$
    else
        Initialize $\mathcal{T}_t \leftarrow \varnothing$
        for each candidate c in the list **do**
            $H' \leftarrow H_t \oplus c$
            Execute $\mathcal{M}[H']$ on $\mathcal{D}_{\text{dis}}$ and add its trajectories to $\mathcal{T}_t$
            Compare H' with H_t on the same tasks
        if any candidate meets the acceptance criterion **then**
            Incorporate the candidate with the largest effectiveness gain into H_t to obtain H_{t+1}
    if $t = T-1$ **then**
        break
Step 2: Failure diagnosis
    $\mathcal{F}_t \leftarrow \text{Diagnose}(\mathcal{T}_t)$: the LLM structures failures as tuples and consolidates diagnoses supported by a shared root cause
Step 3: Constraint generation
    $\mathcal{C}_t \leftarrow \text{Generate}(H_{t+1}, \mathcal{F}_t)$
    Map each proposed change to the node, edge, or graph level according to its target
    Clear the candidate list
    for each candidate $c \in \mathcal{C}_t$ **do**
        $H' \leftarrow H_{t+1} \oplus c$
        if H' fails the syntax and semantics check **then**
            Discard c and **continue**
        Compare H' with H_{t+1} on $\mathcal{D}_{\text{small}}$
        if $\Delta_{\mathcal{D}_{\text{small}}}(H', H_{t+1}) < \theta_{\text{eff}}$ **then**
            Discard c and **continue**
        Compare H' with H_{t+1} on $\mathcal{D}_{\text{val}}$
        if $\Delta_{\mathcal{D}_{\text{val}}}(H', H_{t+1}) < \theta_{\text{gen}}$ **then**
            Discard c and **continue**
        Retain c in the candidate list for the next round
return H_T

A.2 CONSTRAINT TARGETS AND LEVEL ASSIGNMENT

Following the node–edge–graph decomposition in Section 3, constraint targets are assigned according to their execution scope. Node targets concern an individual agent's execution and evidence use; edge targets concern requests and handoffs between agents; graph targets concern shared requirements and group decisions.

B Implementation Details

B.1 Protocols for organization and filtering ablations

This subsection provides the detailed protocols for Tables 3, 18, and 14.

Common optimization-loop schedule. For the Finance-Agent centralized analyses, each fold uses 20 optimization loops. Loop 1 executes H_0 on the 32 diagnosis tasks to collect trajectories, then performs diagnosis, candidate generation, and filtering; it does not update the parent harness. In loops 2–19, candidates retained from the preceding loop undergo normal execution against the current parent, after which the parent may be updated and the resulting trajectories are used for the next diagnosis and generation step. Loop 20 evaluates candidates retained from loop 19 and may update the parent, after which the selected harness is frozen for testing; no new candidate batch is generated in loop 20. Thus, each fold has 19 candidate-generation opportunities and at most 19 parent updates. Within each candidate pool, all candidates are compared against the same fixed parent.

The filtering stages follow Section 4.3.2. Stage 1 applies the rule-based syntax and semantics checks and requires no MAS task execution. Stage 2 evaluates the parent and candidate once on the same fixed four-task subset of the 32 diagnosis tasks and retains the candidate when its mean gain is at least 0.5 points on the 0–100 scale (0.005 on a $[0, 1]$ scale). Stage 3 evaluates the parent and candidate once on the fold's eight validation tasks and retains the candidate when its mean gain is at least 0.25 points (0.0025 on a $[0, 1]$ scale). Candidates passing Stage 3 enter normal execution in the next loop, where the parent and candidate are evaluated on all 32 diagnosis tasks with two repetitions. This normal execution is not a fourth filtering stage. Among candidates completing normal execution, the candidate with the largest strictly positive mean gain is adopted; if none has a positive gain, the parent is retained. Parent scores are not reused across candidates, filtering stages, or loops. The strictly positive adoption criterion is distinct from the > 1-point criterion used only for the candidate-improvement statistic reported in the analysis.

B.1.1 Input–Output Organization

We compare F, U, O, and R on Finance-Agent under the centralized workflow using five-fold evaluation. All variants start from the same H_0 and use the common 20-optimization-loop schedule, with at most three initially generated constraints per generation loop and the same optimizer-session limits. Each variant independently diagnoses its own execution traces under identical evidence-access permissions.

F organizes input-side diagnosis around available conditions, downstream requirements, and missing conditions. Its output-side protocol uses a coordination hypothesis to determine the node-, edge-, and graph-level targets that should be revised together, and permits a single constraint to span multiple levels.

U retains F's structured diagnosis but removes cross-level constraint generation: each generated constraint is restricted to a single node, edge, or graph level.

O uses free-form diagnosis while retaining F's cross-level generation protocol, and therefore may coordinate revisions across multiple levels within one constraint.

R uses free-form diagnosis and single-level generation, restricting each generated constraint to one node, edge, or graph level.

Across all four variants, node-, edge-, and graph-level targets remain available during optimization; the controlled difference is whether multiple levels may be revised jointly within a single generated constraint. All variants use the same three-stage filtering protocol and next-loop normal evaluation and adoption procedure.

B.1.2 Complete failure diagnosis strategies

Using the corresponding runs from the same organization study as the F/U/O/R comparison, we compare complete F/R/L strategies for execution-evidence analysis, revision organization, and candidate implementation requirements. On Finance-Agent under the centralized workflow, all strategies use the same five-fold splits and initial harness H_0. Each uses the common 20-loop

schedule above, with at most three initial candidates per generation loop and the same analysis and generation call allowances. We match search schedules, not realized MAS task starts. Selected harnesses are frozen and evaluated on the held-out folds, with scores averaged over three test repetitions.

F (MASH) analyzes existing conditions, downstream requirements, and unmet dependencies, and uses coordination hypotheses to generate constraints that may jointly revise multiple levels. R uses free-form evidence analysis and single-level free-form constraint generation, as defined in Appendix B.1.1. L organizes both problem analysis and constraint generation separately by node, edge, and graph, with each generated constraint restricted to the level under analysis.

All groups share the MAS configuration, raw-evidence access, underlying harness interfaces, basic implementation-validity checks, the same three-stage filtering protocol, and the same next-loop normal evaluation and adoption procedure. Their diagnosis organization and constraint-composition rules follow their respective strategy definitions above. All strategies may revise multiple objects or hooks within the level scope permitted by that strategy. Only F permits a single generated constraint to jointly revise targets across multiple levels. R restricts each generated constraint to a single level, while L performs diagnosis and constraint generation separately by node, edge, and graph, without coordinating revisions across levels within a single constraint. L need not generate a candidate for every level. In each strategy, the parent harness P remains fixed within a candidate pool, so its candidates are compared against the same parent. Plan-specific consistency checks apply only to F.

We average MAS task starts and optimizer tokens over all 100 optimization loops across the five folds. Each loop covers the applicable analysis, generation and repair, evaluation, and selection steps; loops without an accepted update are included. Initialization and final testing are excluded.

B.1.3 Filtering-stage ablations

We compare Full MASH with three stage-level removals on Finance-Agent under the centralized workflow using the five-fold splits in Appendix B.3. Within each fold, all four configurations independently use the same H_0, common 20-loop schedule, diagnosis and generation procedure, editable scope, and normal next-loop evaluation and adoption rule. The optimizer is GPT-5.4, with at most three initial candidates per generation loop and session caps of 11 for analysis and 13 for generation. We match search schedules rather than realized LLM request counts.

- **w/o Stage 1** removes the explicit rule-based syntax-and-semantics filtering and its associated interception, repair, and feedback. F's generation requirements remain unchanged; runtime failures are still recorded under the common execution-failure handling.
- **w/o Stage 2** removes the fixed four-task effectiveness executions, the 0.5-point threshold decision, and the associated filtering feedback. Stage-1 survivors proceed directly to Stage 3.
- **w/o Stage 3** removes the eight-task robustness executions, the 0.25-point threshold decision, and the associated filtering feedback. Stage-2 survivors proceed directly to normal execution in the next loop.

All remaining filtering stages, the subsequent 32-task two-repetition normal execution, and the strictly positive parent-adoption rule are unchanged.

After search, the selected harness in each fold is frozen. For each of three test repetitions, we compute the mean score over the 50 held-out tasks pooled across the five folds. We report the mean of these three repetition-level scores.

Search cost includes all LLM requests for diagnosis, generation and repair, and MAS execution, including requests from failed attempts and retries. We average search requests over all 100 optimization loops across the five folds, excluding initialization and final testing.

B.2 Protocols for generalization experiments

B.2.1 Generalization across workflows

We independently repeat harness optimization under centralized, decentralized, hybrid, and independent topologies. The admissible decision points automatically contract to those exposed by each fixed workflow, without changing its topology or introducing new communication channels. In the independent workflow, edits are restricted to local execution and the existing terminal aggregation, with no intermediate inter-agent communication or shared intermediate agent state.

B.2.2 Backbone transfer and adaptation

We evaluate transfer and adaptation on Finance-Agent under the centralized workflow, using Gemini 2.5 Flash-Lite as the source execution backbone and GPT-4o mini and Claude Haiku 4.5 as targets.

For each target backbone, we compare three conditions. *Base MAS* uses the initial harness H_0. *Frozen transfer* applies the source-optimized harness without modification or target-side search. *Re-optimized* continues optimization under the target backbone, starting from the transferred harness.

The MAS roles, topology, tool permissions, scorer, per-task execution budgets, and diagnosis and harness-generation models remain fixed, as do the common search rules. For *Re-optimized*, both targets receive 20 additional search rounds, each allowing at most three initial candidates, with session caps of 11 for analysis and 13 for generation. Within each Finance-Agent fold, transfer uses the source harness selected on that fold's development partition. Target-side adaptation uses only the same fold's discovery and validation partitions.

B.3 Data splits

Finance-Agent uses five-fold outer evaluation on its 50 public tasks, with 32 discovery, 8 validation, and 10 test tasks per fold and independent optimization and selection in each fold. For the Finance-Agent centralized analyses, the Stage-2 effectiveness screen uses a fixed four-task subset of the 32 discovery tasks, selected before search and shared across applicable variants and optimization loops within the fold. This subset is not an additional data partition. The eight validation tasks are used by the Stage-3 robustness check, while the 10 test tasks remain withheld until the selected harness is frozen.

The other benchmarks use 80 discovery, 20 validation, and 100 test tasks each. Partitions are fixed and shared across methods within each benchmark. The four-task screening subset and the numerical Stage-2/Stage-3 thresholds specified in Appendix B.1 apply to the Finance-Agent centralized analyses and are not asserted for the other benchmarks.

B.4 Baseline Details and Adaptation Protocols

We group the evaluated baselines according to their primary optimization scope: a static MAS reference, automatic harness/context optimization methods, and MAS/workflow optimization methods. For each method, we preserve its native optimization and selection mechanism as far as it is compatible with the controlled MAS setting used in the main comparison.

Common adaptation protocol. Within each benchmark, all baselines operate on the same initial MAS configuration and use the same agent roles and count, communication topology, task-execution backbone, tool permissions, scorer, data partitions, and per-task execution limits. We distinguish the fixed MAS structure from the runtime behavior implemented within that structure: agent membership, roles, and communication edges remain fixed, while methods may optimize prompts, state handling, coordination policies, aggregation, or control logic when these components fall within their native optimization scope. The task-execution backbone is Gemini 2.5 Flash-Lite with temperature 0.0. All optimization methods share a development task-attempt cap within each benchmark, while retaining method-specific optimization and selection procedures. The Finance-Agent budget, schedules, and compute accounting are specified below. When a method natively supports edits outside this controlled boundary, we disable those structural edits while retaining its native optimization, feedback, and

Table 4: Finance-Agent search schedules per fold, subject to the common 4,000-task-attempt cap and method-specific stopping criteria.

Method	Search schedule
GEPA	Three-task reflection minibatches and eight-task validation; budget-aware termination.
RHI	At most 20 shared-harness revisions; 32 tasks once per revision; parent results reused.
ACE	Up to five passes over 32 tasks; at most five correction steps per sample; per-sample curation.
AHE	Ten evaluated artifacts: the initial version and nine revisions; 32 tasks with two repeats.
Life-Harness	At most 20 new revisions beyond the initial artifact; 32 tasks with two repeats.
Meta-Harness	At most 20 proposer sessions, with up to three initial candidates each; 32 tasks with two repeats.
HarnessX	At most 20 rounds and four candidate slots per round; fresh parent evaluation; 32 tasks with two repeats.
Meta-Team	One pass over 32 tasks; evolution after each original task; at most one redo without retriggering evolution.
AFlow	At most 20 tree expansions; 32 tasks with five repeats per artifact; native convergence stopping.
MASH	Twenty cycles, including at most 19 generation cycles with three initial candidate slots each; the final cycle evaluates pending candidates.

selection procedures wherever applicable. After optimization, the resulting artifact is frozen before evaluation on $\mathcal{D}_{\text{test}}$.

Finance-Agent budget and stopping rules. For Finance-Agent under the centralized workflow, we use five folds with 32 discovery, eight validation, and ten test tasks per fold. The development cap is 4,000 MAS task attempts per method per fold, with no transfer of unused capacity between folds. Each launched task-level execution counts once, including initialization, fresh parent executions, candidate evaluations, development-set validation, failed executions, and whole-task retries. Reused results incur no new attempt; final testing is excluded. Methods stop at their configured schedule limit, native stopping criterion, or when they cannot continue within the remaining budget. Table 4 specifies the schedules. The common cap bounds MAS execution opportunities, while allowing method-specific consumption and optimizer-side compute.

All optimizer-side roles in this Finance-Agent configuration use GPT-5.4, including reflection, diagnosis, generation, repair, and internal judging. Coding-agent subrequests are included. ACE's task-solving generator remains part of the Gemini-based executor; its reflector and curator belong to the optimizer. The benchmark scorer is accounted for separately.

GEPA starts a mutation only when the remaining budget can cover its 14-attempt normal evaluation path: three parent executions, three candidate executions, and eight validation executions. Reserved but unlaunched executions are not counted. HarnessX retains its early-stopping criterion and may terminate before the round limit with budget remaining. The final artifact is frozen after development-set selection, without an additional shared shortlist, and each test task is evaluated three times.

Static reference. **MAS-Centralized** (Kim et al., 2026) instantiates the centralized MAS architecture under our common backbone, tool, and execution-budget configuration and serves as the unoptimized reference. It retains the initial agent instructions, communication interfaces, and orchestration logic without additional harness optimization.

Automatic harness and context optimization. **GEPA** (Agrawal et al., 2026) is a reflective prompt optimizer that uses execution trajectories and natural-language reflection to diagnose failures, propose prompt revisions, evaluate them, and retain complementary candidates through Pareto-based selection. In our adaptation, GEPA edits the natural-language instructions exposed through the existing agent and orchestration interfaces of the fixed MAS. Agent membership, communication edges, tools, and executable orchestration structure remain unchanged. Thus, GEPA retains its trajectory-driven reflective prompt evolution while serving as a prompt-level optimization baseline.

RHI (Lee et al., 2026a) represents an agent harness as a prompt-level specification of the agent loop and recursively refines that specification using pairwise feedback accumulated over its own revision history. The specification can describe agent instructions, communication contracts, and textual interaction policies. For the fixed-MAS comparison, we retain RHI's pairwise-history refinement procedure but restrict revisions to textual specifications that are executable within the existing MAS. Revisions may refine role instructions and information-exchange contracts, but may not add or remove agents, communication links, or tools, or otherwise change the fixed communication topology.

ACE (Zhang et al., 2025d) treats context as an evolving playbook and improves it through generation, reflection, and incremental curation. Rather than modifying model weights, ACE accumulates and organizes task-relevant strategies in context. We retain this playbook-evolution procedure and expose the resulting playbook through the existing context interfaces of the fixed MAS. ACE does not modify the executable MAS topology, tool permissions, or environment. The final curated playbook produced on the development partitions is frozen before test evaluation.

AHE (Lin et al., 2026) performs automatic harness engineering over explicit editable components and uses execution evidence to guide and verify successive edits. Its original design exposes harness components such as tool-related logic, middleware, and long-term memory as observable and revertible optimization targets. In our adaptation, AHE retains its evidence-driven component-editing loop, while the admissible edit surface is restricted to components compatible with the fixed MAS interfaces. It may revise harness-side descriptions, state or memory logic, and middleware behavior exposed by the implementation, but it cannot add new external tools, expand tool permissions, change the task environment, or alter the fixed agent topology.

Life-Harness (Xu et al., 2026b) adapts the runtime interface around a frozen agent by turning recurring execution failures into reusable interventions over environment contracts, procedural skills, action realization, and trajectory regulation. For the MAS setting, these interventions are instantiated through the existing agent and orchestration interfaces rather than by introducing new agents or communication channels. We retain these classes of runtime intervention while keeping the benchmark environment, task-execution model, available tools, and their external semantics fixed. The learned interventions therefore modify runtime behavior within the existing MAS rather than changing the underlying task environment or communication structure.

Meta-Harness (Lee et al., 2026b) searches directly over harness code. Its proposer can inspect source code, candidate scores, and execution traces from previous attempts to diagnose failures and propose code-level harness revisions. We preserve this outer-loop code-search mechanism and its access to development execution evidence. Candidate edits are restricted to the harness-side code exposed by the fixed MAS implementation. Edits that would change the task-execution model, agent count, communication topology, external tool set, or tool permissions are outside the admissible search space.

HarnessX (Chen et al., 2026b) models the runtime harness as composable typed primitives spanning prompts, tools, memory, and control flow, and adapts their composition using trace-driven evolution. For our comparison, we retain its primitive-based composition and trace-driven adaptation while restricting substitutions and recompositions to primitives compatible with the fixed MAS interfaces and topology. The method may adapt harness behavior within existing agent and orchestration interfaces, but cannot introduce new agents, communication channels, external tools, or additional tool permissions.

MAS and workflow optimization. **Meta-Team** (Hao et al., 2026) is designed for experience-driven evolution of multi-agent systems, including agent behavior, inter-agent coordination, and team-level organization. Because the main comparison fixes agent membership, roles, and communication topology, we use a fixed-structure adaptation of Meta-Team. Its experience aggregation and collaborative evolution procedures are retained, while structural edits that would add or remove members, change fixed roles, or alter the fixed team or communication structure are disabled. The remaining optimization surface covers behavioral instructions, coordination policies, and team-level execution rules realizable on the existing MAS graph. Thus, the adaptation preserves experience-driven multi-level improvement within the fixed MAS structure while excluding structural team reconfiguration.

AFlow (Zhang et al., 2025c) formulates agentic workflow optimization as search over code-represented workflows whose LLM-calling nodes are connected by edges. It uses Monte Carlo

Table 5: Development compute on Finance-Agent under the centralized workflow, aggregated over five folds. Each optimization method is subject to a cap of 4,000 MAS task attempts per fold, enforced independently within each fold; unused capacity is not transferred between folds. Input and output denote optimizer tokens in millions. Final testing is excluded.

Method	Task attempts	Optimizer calls	Input (M)	Output (M)
MAS-Centralized	0	0	0.00	0.00
GEPA	19,742	2,147	26.31	3.41
RHI	3,487	3,287	18.94	1.34
ACE	3,118	2,241	41.76	2.07
AHE	3,276	12,083	151.37	12.64
Life-Harness	6,914	873	17.68	1.37
Meta-Harness	17,284	2,317	58.42	4.71
HarnessX	17,533	2,286	48.11	4.18
Meta-Team	236	2,296	38.87	1.84
AFlow	13,684	131	2.71	0.34
MASH	16,946	913	8.27	0.93

Tree Search, tree-structured experience, code modification, and execution feedback to iteratively improve workflow candidates. Because unrestricted AFlow can modify workflow structure, we retain its search, experience, and execution-feedback mechanisms while restricting candidate workflows to the existing agents and communication edges. Within this fixed graph, AFlow may optimize compatible orchestration logic, node instructions, execution scheduling subject to the fixed communication dependencies, and aggregation or verification logic. This constitutes a fixed-structure adaptation of AFlow's workflow optimization procedure rather than unrestricted workflow-topology search.

Compute accounting. We aggregate development-time compute from task-execution records and optimizer request logs. Each launched task-level MAS execution counts as one task attempt, including initialization, fresh parent executions, candidate evaluations, development-set validation, failed executions, and whole-task retries. Reusing a previously completed execution incurs no additional attempt, and final test executions are excluded.

Optimizer calls count individual model requests for analysis, reflection, diagnosis, generation, repair, internal judging, and subsidiary optimization agents, including retries whenever a new model request is actually issued. An optimization session may therefore contain multiple optimizer calls. Tool invocations themselves are not counted as optimizer calls, while restarting an entire MAS task contributes an additional task attempt.

Input and output token totals are obtained by summing request-level usage reported by the corresponding model APIs. Provider-specific cached-input and reasoning-token fields are treated as breakdowns of the reported usage when applicable and are not added again to the corresponding token totals. We apply these accounting definitions consistently when normalizing the execution records of all methods. MAS-internal model calls and benchmark-grader usage are accounted for separately from optimizer usage.

Table 5 separates the workload of executing the MAS from the additional model usage introduced by the optimizer. The common task-attempt cap therefore defines a shared upper bound on MAS evaluation opportunities rather than requiring identical realized compute across methods.

The observed profiles reflect substantial differences in how the optimization procedures allocate computation. AHE uses relatively few task-level MAS executions but issues many optimizer requests through its task-level debugging and refinement procedure. AFlow shows the opposite pattern, evaluating many MAS executions for each optimizer-side request. Meta-Team performs approximately one development pass over the tasks in each fold, with additional executions arising from retries and auxiliary evaluation, while its per-task evolution procedure produces substantially more optimizer requests than MAS task attempts. GEPA operates close to the per-fold MAS-execution cap, whereas several methods terminate earlier because of their native search schedules or stopping criteria.

MAS-Centralized performs no development-time optimization and therefore incurs no development optimization cost in this table; its frozen test executions are excluded under the same reporting

convention. The quantities in Table 5 should not be interpreted as a ranking of total compute, because a task attempt does not capture the amount of model computation performed inside an MAS execution. Instead, the table reports the two principal resource axes separately: task-level MAS execution and optimizer-side model usage.

B.5 Failure Labeling Protocol

Scope and unit of analysis. This protocol defines execution-level labeling of node-, edge-, and graph-level issues in frozen-harness test trajectories. It measures observable process defects, not their causal contributions to task failure. For Finance-Agent, the evaluation design comprises five held-out folds covering 50 tasks, with three repetitions per task and 150 executions per method. BrowseComp-Plus, PlanCraft, and WorkBench are evaluated on their respective held-out test sets, also with three repetitions per task. Only executions receiving less than the maximum task score enter the labeling population. Each execution is identified by its method, benchmark, task, repetition, and, for Finance-Agent, fold. Method identity remains on the analysis side and is hidden from annotators. Labels are assigned directly from trajectories, independently of search-time diagnoses and other error taxonomies.

Evidence available to annotators. The annotation input consists of the task requirements, member roles, assigned requests, member-visible model inputs, tool observations, delivered contributions, and team decisions, ordered by event identifiers. Method names, task scores, reference answers, rubric judgments, and labels from other annotators are excluded. The external score determines eligibility only. Evidence records must identify the relevant events and preserve their surrounding context. A claim that information was omitted from a recipient's input requires inspection of the relevant input snapshot, including other visible messages and context blocks; a short quotation alone does not establish absence.

Operational definitions. The labeling criterion is the responsibility of the defective decision, not the identity of the acting agent, the file containing its implementation, or the hook through which it could be repaired.

- **Node.** A member's local retrieval, extraction, reasoning, calculation, tool use, or verification fails to satisfy its assigned work under the inputs available to that member. Examples include using an incorrect unit despite receiving the correct unit, miscomputing from available values, or selecting incorrect tool arguments. An unsuccessful search or a tool-service failure alone does not establish a node issue.
- **Edge.** An assigned request or a delivered contribution fails to preserve information required for the interaction. Examples include omitting an essential constraint from an assigned request or dropping an already available reporting period or source binding during delivery. Information that the sender never acquired is not, by itself, a delivery omission. Failure to allocate a necessary task is a graph issue rather than a request-expression issue.
- **Graph.** Team-level allocation, acceptance, aggregation, conflict resolution, coverage checking, or stopping fails to satisfy the requirements of the overall task. Examples include leaving necessary work unassigned, failing to integrate an available contribution into a team decision, or ending while required work remains unresolved. A prescribed budget stop does not by itself establish a defective stopping decision.

Label states and overlap. Each level receives one of three labels: *present*, *absent*, or *unknown*. Present requires a specific responsibility defect supported by locatable evidence. Absent means that the relevant process is inspectable and no such defect is found; it does not establish that the execution is otherwise correct. Unknown applies when missing inputs, incomplete logs, or ambiguous evidence prevent a decision. An incorrect final answer is not sufficient evidence for any level. Infrastructure errors are recorded separately and do not automatically receive a process-error label.

Multiple levels may be present in one execution, but each requires its own supporting evidence. An upstream calculation error transmitted unchanged is not automatically an additional edge issue, and a wrong aggregate answer is not automatically a graph issue. Conversely, an independently evidenced delivery defect and a defective team decision may both be labeled. The protocol neither forces a

Table 6: Inter-annotator agreement for node-, edge-, and graph-level failure labeling. Executions denotes the number of eligible non-perfect executions receiving independent secondary annotation, and judgments counts the corresponding level-wise decisions across the three levels. Agreement and unweighted Cohen's κ are computed from the pre-adjudication three-class labels (*present*, *absent*, and *unknown*).

Benchmark	Executions	Judgments	Agreement	Cohen's κ
Finance-Agent	62	186	92.5%	0.85
BrowseComp-Plus	135	405	90.1%	0.81
PlanCraft	77	231	90.9%	0.83
WorkBench	105	315	90.5%	0.83

single root-cause label nor duplicates one observed defect across levels solely because its effects propagate.

Annotation and review procedure. The primary annotator role is assigned to `gpt-5.5`, and the independent secondary role to `gemini-2.5-flash`. Both receive the same decision rules and evidence fields. The operative instruction is to assess each level independently, distinguish missing evidence from evidence of absence, and support every present label with a responsibility-specific reason and event references. Required output fields are the anonymous execution identifier and, for each level, its label, reason, and evidence references. The annotation record also retains the requested model identifier, returned identifier when available, prompt version, and explicit decoding settings.

For independent review, task identifiers are selected before inspecting the primary labels. For Finance-Agent, two held-out task identifiers are sampled from each of the five folds, yielding 10 task identifiers in total. For BrowseComp-Plus, PlanCraft, and WorkBench, 20 held-out task identifiers are sampled independently for each benchmark. Within each benchmark, the same sampled task identifiers define the review frame for MAS-Centralized, Meta-Harness, and MASH, and for all three test repetitions. Only eligible non-perfect executions within this frame undergo secondary labeling.

The secondary annotator cannot inspect the primary output. Disagreements are adjudicated against the original trajectory evidence, with unknown retained when the available evidence cannot resolve the issue. Ten additional executions are randomly sampled from records with complete three-level agreement for evidence and boundary checks. Raw labels, adjudications, and final labels remain separate. Agreement and unweighted three-category Cohen's κ are calculated from the pre-adjudication labels, with *unknown* retained as a separate category. Final failure-count analyses use the adjudicated labels.

Aggregation. For method m, let D_m be its non-perfect execution set and let $y_{i,\ell}$ be the final label of execution i at level ℓ. The confirmed issue count and its within-population percentage are

$$C_{m,\ell} = \sum_{i \in D_m} \mathbf{1}[y_{i,\ell} = \text{present}], \qquad P_{m,\ell} = 100\,\frac{C_{m,\ell}}{|D_m|}.$$

Repeated occurrences at one level within an execution count once. The coordination union is computed by execution identity:

$$C_{m,E\cup G} = \sum_{i \in D_m} \mathbf{1}[y_{i,E} = \text{present} \ \vee \ y_{i,G} = \text{present}].$$

Unknown labels are not converted to absent. The within-method percentages retain all non-perfect executions in the denominator and therefore quantify confirmed issue occurrence within the non-perfect execution population, rather than prevalence among only assessable records. For every method, benchmark, and level, the three label states exhaust the corresponding non-perfect execution set:

$$N^{\text{present}}_{m,\ell} + N^{\text{absent}}_{m,\ell} + N^{\text{unknown}}_{m,\ell} = |D_m|.$$

Executions with no present label are separated into those with three absent labels and those containing at least one unknown label. Between-method count differences do not identify paired repairs, and the overlapping level counts cannot be summed to explain the number of additional successful executions.

Table 7: Unknown labels in the execution-level failure analysis. Non-perfect executions are reported for MAS-Centralized / Meta-Harness / MASH. Each unknown entry reports Node / Edge / Graph counts. Unknown labels remain distinct from absent labels and remain in the denominator of the corresponding within-method issue percentages.

Benchmark	Non-perfect executions	MAS-Centralized unknown	Meta-Harness unknown	MASH unknown
Finance-Agent	117 / 101 / 91	4 / 2 / 2	3 / 2 / 1	3 / 1 / 1
BrowseComp-Plus	240 / 225 / 213	8 / 10 / 7	7 / 9 / 7	7 / 9 / 6
PlanCraft	141 / 131 / 115	3 / 5 / 4	3 / 4 / 4	3 / 4 / 3
WorkBench	195 / 173 / 156	6 / 7 / 6	5 / 6 / 5	5 / 6 / 5

Table 8: Confirmed issues in non-perfect Finance-Agent test executions under the centralized workflow. Issue rows report the confirmed issue count followed by its percentage relative to that method's non-perfect executions. Each execution is counted at most once per level; level labels may overlap, and the edge-or-graph union is deduplicated by execution. Unknown labels remain in the corresponding non-perfect-execution denominator and are reported separately in Appendix B.5.

Metric	MAS-Centralized	Meta-Harness	MASH
Non-perfect executions	117	101	91
Node	56 (47.9%)	36 (35.6%)	32 (35.2%)
Edge	42 (35.9%)	32 (31.7%)	20 (22.0%)
Graph	45 (38.5%)	36 (35.6%)	21 (23.1%)
Edge or graph (union)	66 (56.4%)	49 (48.5%)	31 (34.1%)

C Additional Results

C.1 Collaboration error analysis

We analyze frozen-harness test trajectories under the centralized workflow using the execution-level labeling protocol in Appendix B.5. Each non-perfect execution is labeled independently for node-, edge-, and graph-level issues from trajectory evidence. The labels describe observable process defects rather than their causal contributions to task failure, and multiple levels may be present in the same execution.

For Finance-Agent, the five held-out folds contain 50 test tasks in total. With three test repetitions per task, each method produces 150 frozen-harness test executions. Table 8 reports the detailed Finance-Agent breakdown.

Meta-Harness reduces the confirmed node-level issue count relative to the initial MAS, while coordination-related issues remain frequent. Relative to Meta-Harness, MASH further reduces the confirmed Finance-Agent issue counts from 36 to 32 at the node level, from 32 to 20 at the edge level, and from 36 to 21 at the graph level. These correspond to relative reductions in confirmed issue counts of 11.1%, 37.5%, and 41.7%, respectively. The deduplicated edge-or-graph union also decreases from 49 to 31 executions.

The percentages in Table 8 and the relative reductions reported above have different denominators. The table percentages are within-method quantities, $C_{m,\ell}/|D_m|$, where D_m is that method's non-perfect execution set. In contrast, the Meta-Harness-to-MASH reductions compare confirmed issue counts directly between the two methods. Neither statistic is interpreted as a causal contribution of the corresponding issue type to task performance.

For benchmark b and level ℓ, let $C_{b,\ell}^{\mathrm{Meta}}$ and $C_{b,\ell}^{\mathrm{MASH}}$ denote the confirmed issue counts under Meta-Harness and MASH, respectively. We compute the relative reduction in confirmed issue count as

$$r_{b,\ell} = \frac{C_{b,\ell}^{\mathrm{Meta}} - C_{b,\ell}^{\mathrm{MASH}}}{C_{b,\ell}^{\mathrm{Meta}}} \times 100\%.$$

Table 9: Reduction in confirmed node-, edge-, and graph-level issue counts from Meta-Harness to MASH across four benchmarks under the centralized workflow. Each benchmark cell reports the confirmed issue counts Meta-Harness → MASH, followed by the relative reduction in confirmed issue count. The final row reports the unweighted macro-average of the four benchmark-level relative reductions.

Benchmark	Node	Edge	Graph
Finance-Agent	36 → 32 (11.1%)	32 → 20 (37.5%)	36 → 21 (41.7%)
BrowseComp-Plus	113 → 108 (4.4%)	78 → 56 (28.2%)	74 → 55 (25.7%)
PlanCraft	85 → 77 (9.4%)	35 → 32 (8.6%)	65 → 50 (23.1%)
WorkBench	87 → 82 (5.7%)	54 → 39 (27.8%)	89 → 66 (25.8%)
Macro average	**7.7%**	**25.5%**	**29.1%**

Table 10: Generalization across fixed workflows. Scores are multiplied by 100 and averaged over three test repetitions with each selected harness frozen. Centralized results are reproduced from Table 1. Best baseline denotes the highest reported baseline score for each benchmark–workflow pair. Parentheses identify the baseline: HX = HarnessX, MH = Meta-Harness, and AF = AFlow; multiple names indicate a tie. Best scores are bold. Improvement is MASH minus the best baseline in absolute score points.

Method	Finance-Agent	BrowseComp-Plus	PlanCraft	WorkBench
Centralized				
MAS-Centralized	39.97	20.00	53.00	35.00
Best baseline	48.76 (HX)	25.33 (HX)	57.00 (AF)	43.33 (HX)
MASH	**52.88**	**29.00**	**61.67**	**48.00**
Improvement	+4.12	+3.67	+4.67	+4.67
Decentralized				
MAS-Decentralized	38.26	23.67	53.00	41.00
Best baseline	46.25 (HX)	29.33 (MH/AF)	56.67 (AF)	49.33 (HX)
MASH	**48.97**	**34.67**	**60.33**	**53.33**
Improvement	+2.72	+5.34	+3.66	+4.00
Hybrid				
MAS-Hybrid	39.55	21.67	55.00	37.00
Best baseline	47.86 (HX)	28.00 (HX)	59.67 (AF)	44.33 (MH)
MASH	**50.76**	**32.33**	**63.67**	**49.67**
Improvement	+2.90	+4.33	+4.00	+5.34
Independent				
MAS-Independent	35.75	14.67	45.00	46.00
Best baseline	40.71 (HX)	18.67 (HX)	47.33 (HX/AF)	48.67 (HX)
MASH	**42.43**	**24.00**	**52.67**	**52.00**
Improvement	+1.72	+5.33	+5.34	+3.33

For each level, we then report the unweighted macro-average

$$\bar{r}_\ell = \frac{1}{4} \sum_b r_{b,\ell}.$$

Table 9 yields macro-averaged relative reductions in confirmed issue counts of 7.7%, 25.5%, and 29.1% for node-, edge-, and graph-level issues, respectively. The larger reductions observed for edge- and graph-level issue counts than for node-level issue counts are consistent with MASH addressing inter-agent information exchange and team-level coordination in addition to individual-agent execution. These count reductions should not be interpreted as paired repairs of the same executions or as estimates of the causal effect of each issue type on final task performance.

C.2 Generalization across workflows

Table 10 reports the full mean scores of the initial MAS, the best reported baseline, and MASH for each benchmark–workflow pair.

Table 11: Candidate quality and acceptance on Finance-Agent under the centralized workflow. Counts are pooled across five folds. Each entry reports numerator/denominator (percentage). The largest rate in each column is bold.

Variant	Qualification Q/N	Improvement I/M	Acceptance A/N
F (MASH)	96/164 (**58.5%**)	56/87 (**64.4%**)	29/164 (17.7%)
U	94/173 (54.3%)	51/84 (60.7%)	38/173 (**22.0%**)
O	86/169 (50.9%)	43/78 (55.1%)	30/169 (17.8%)
R	84/181 (46.4%)	36/74 (48.6%)	29/181 (16.0%)
L	94/176 (53.4%)	49/84 (58.3%)	33/176 (18.8%)

Table 12: Development evaluation cost and cascade savings on Finance-Agent under the centralized workflow, pooled across five folds. N counts submitted versions; S counts scheduled evaluations avoided by normal filtering rejection. The avoided fraction is $S/(152N)$.

Variant	N	Development task starts	Avoided evaluations S	Avoided (%)
F (MASH)	164	16,946	9,640	38.7
U	173	15,820	11,160	42.4
O	169	15,191	11,768	45.8
R	181	14,868	13,704	49.8
L	176	16,292	11,600	43.4

C.3 Candidate quality and evaluation cost

These analyses reuse the Finance-Agent centralized-workflow records from the organization study for F, U, O, R, and L, under the common search schedule and optimizer-session limits. Tables 11 and 12 provide the complete counts and rates underlying Figure 3.

Candidate quality and acceptance. For each variant, N counts submitted candidate harness versions, including initial implementations and versions submitted after legality repair or quality revision; log entries are deduplicated by version ID. Q counts versions passing all three filtering stages under the current protocol, and A counts versions selected at round end to update the parent harness. The qualification and acceptance rates are Q/N and A/N. M counts versions with complete parent–candidate evaluations on the same 32 discovery tasks, with two repetitions per task. Because this normal evaluation occurs after filtering, Q may exceed M when a filtering-qualified version does not complete the subsequent normal evaluation. The corresponding $Q - M$ counts for F/U/O/R/L are 9, 10, 8, 10, and 10, respectively. Among the M completed versions, I counts those with $\bar{\Delta}_{\text{dis}} > 0.01$, a gain of more than one point on the 100-point scale. We report the improvement rate I/M. All rates are calculated from counts pooled across the five folds.

Evaluation cost and cascade savings. Under the current protocol, a submitted candidate that reaches all execution-based components has a complete scheduled allocation of 152 MAS task starts: 8 for the Stage-2 four-task effectiveness screen, 16 for the Stage-3 eight-task robustness check, and 128 for the subsequent normal evaluation on 32 diagnosis tasks with two repetitions for both parent and candidate. Stage 1 requires no MAS task execution. S counts scheduled task starts avoided by normal filtering rejection. Using the same submitted-version count N, we compute the avoided fraction $S/(152N)$ from counts pooled across the five folds. Actual development MAS task starts are reported separately and also include initialization, separate parent-only executions, one-sided starts from incomplete pairs, and whole-task retries, while excluding final testing.

Reconciliation of scheduled evaluations and task starts. The remaining scheduled allocation $152N - S$ is not itself the number of actual development task starts. Let D_{inc} count planned slots absent from completed parent–candidate pair records because post-filtering normal evaluations remained unfinished. These slots are not included in S. An incomplete pair does not imply that neither side was launched: C_{unp} adds back the one-sided starts that occurred but are not represented by completed pairs. Let C_{parent} count first launches from separate parent-only executions used to obtain diagnosis trajectories when no candidate proceeds to normal evaluation. Initialization starts

C_{init} include retries occurring during initialization. Additional whole-task retries are counted in C_{retry}; this term excludes initialization retries and includes retries of separate parent-only executions, avoiding double counting. The development total is

$$C_{\text{dev}} = 152N - S - D_{\text{inc}} + C_{\text{unp}} + C_{\text{parent}} + C_{\text{init}} + C_{\text{retry}}. \quad (1)$$

Table 13 gives the complete reconciliation. Unfinished evaluations do not increase S, so the normal-filtering avoided fraction remains $S/(152N)$.

Table 13: Reconciliation of development MAS task starts, pooled across the five Finance-Agent folds. Minus and plus signs indicate the operations in Equation 1. Residual is the reconciled total minus the reported total in Table 12.

Accounting item	F	U	O	R	L
Complete schedule $152N$	24,928	26,296	25,688	27,512	26,752
Avoided by filtering $(-S)$	9,640	11,160	11,768	13,704	11,600
Remaining allocation $152N - S$	15,288	15,136	13,920	13,808	15,152
Incomplete slots $(-D_{\text{inc}})$	432	648	556	512	382
Unpaired starts $(+C_{\text{unp}})$	9	10	8	10	10
Separate parent-only starts $(+C_{\text{parent}})$	1,664	896	1,408	1,152	1,088
Initialization $(+C_{\text{init}})$	320	322	322	328	320
Whole-task retries $(+C_{\text{retry}})$	97	104	89	82	104
Development starts C_{dev}	16,946	15,820	15,191	14,868	16,292
Residual	0	0	0	0	0

All five reconciled totals agree with Table 12. The incomplete-slot and unpaired-start rows are paired accounting terms: D_{inc} removes all slots missing from completed pairs, while C_{unp} adds back one-sided starts that actually occurred. Their net unstarted counts are 423, 638, 548, 502, and 372 for F/U/O/R/L, respectively. Per-loop resource summaries derived from this ledger are reported separately in Appendix C.8; those summaries exclude initialization and final testing.

Analysis on constraint filtering strategies. Table 14 shows that the filtering stages play complementary roles. Removing syntax-and-semantics or effectiveness filtering increases LLM calls by 9.7% and 12.3%, respectively, while reducing final performance, indicating that both stages avoid unnecessary candidate execution. Removing robustness filtering saves only 1.7% of calls but yields the largest performance drop, suggesting that its modest additional cost is justified by improved constraint quality. Together, the three stages balance evaluation cost and final harness performance.

C.4 Analysis of cross-level constraint generation

We obtain a common node-strengthened harness H_N through 20 rounds of node-only optimization. Each variant then starts from H_N for 20 additional rounds. A permits node-level revisions only; B freezes node-level rules and permits edge- and graph-level revisions; C permits revisions at all three levels. Frozen rules remain active during execution. All variants use GPT-5.4 as the optimizer and the same evaluation rules, with at most three initial candidates per round and session caps of 11 for analysis and 13 for generation.

Table 15 shows that all-level continuation achieves the highest observed score on all four benchmarks. Its advantage over node-only continuation is larger on Finance-Agent and BrowseComp-Plus, but small on PlanCraft, indicating that the benefit of expanding the editable levels is task-dependent.

Coordination-only continuation outperforms node-only continuation on Finance-Agent and BrowseComp-Plus, but underperforms it on PlanCraft and WorkBench. Thus, freezing local rules and optimizing coordination alone is not uniformly beneficial. Together, these results support keeping both local and coordination decisions editable during further optimization.

C.5 Distribution of accepted revisions

We analyze the 29 accepted revisions of full MASH reported in Table 11, pooled across the five Finance-Agent folds. Each revision corresponds to one parent-harness update during optimization.

Table 14: Filtering-stage ablations on Finance-Agent (centralized). Scores are mean test scores ($\times 100$). LLM calls are averaged over 100 optimization loops across five folds, including failed calls and retries but excluding initialization and final testing. The last column reports the change in calls relative to Full MASH.

Variant	Score	LLM calls / loop	Call change
Full MASH	**52.88**	13,447.3	–
w/o Syntax/Semantics	52.43	14,757.7	+9.7%
w/o Effectiveness	52.67	15,102.7	+12.3%
w/o Robustness	51.96	13,223.8	−1.7%

Table 15: Editable-level ablation with 20 additional optimization rounds from a common harness H_N obtained after 20 node-only rounds. A permits node-level edits only; B permits edge- and graph-level edits with node rules frozen; C permits edits at all three levels. Scores are multiplied by 100; higher is better. Best scores are bold, and Improvement reports C minus A in absolute score points.

Variant	Finance-Agent	BrowseComp-Plus	PlanCraft	WorkBench
A: Node	48.12	25.00	61.00	45.00
B: Edge + Graph	51.60	28.00	58.33	44.33
C: Node + Edge + Graph	**53.12**	**29.33**	**61.33**	**47.67**
Improvement: C − A	+5.00	+4.33	+0.33	+2.67

At the hook level, we count each modified hook once per accepted revision, yielding 64 hook modifications. At the candidate level, each accepted revision belongs to exactly one of four mutually exclusive categories: node only, edge only, graph only, or cross-level (modifying hooks at two or more levels). Table 16 provides the complete counts and shares underlying Figure 5.

Figure 5(a) shows that graph- and node-level edits account for similar shares of hook modifications, both larger than the edge-level share. Figure 5(b) shows that cross-level revisions form the largest single candidate category. Most accepted revisions modify at least one edge- or graph-level hook, extending beyond the node-only editable scope. The performance consequences of expanding this scope are examined in the admissible-level ablation (Table 15).

C.6 Case Study: Evidence Binding in a Finance Task

We trace a finance task asking how many basis points Micron (MU) beat or missed its Q3 FY2024 GAAP gross margin guidance. A centralized MAS assigned three workers to retrieve the guidance, retrieve the actual result, and calculate the difference. Two workers reported unresolved guidance, while another treated approximately 81% as the guidance and compared it with the actual margin of 26.9%. The lead adopted the resulting answer: a miss of 5,410 basis points.

Table 17: Failure diagnosis on the MU finance case.

Level	Observed failure	Missing information
Edge	The guidance claim reaches the lead without matching evidence.	Source evidence supporting the value as the requested guidance.
Node	A worker selects approximately 81% as Q3 FY2024 GAAP guidance.	Verification of the fiscal period, accounting basis, and role as guidance.
Graph	The lead accepts the comparison despite unresolved guidance reports.	A guidance operand supported by matching evidence.

Tracing the final comparison backward through the handoff reaches the worker's initial selection of the unsupported value. MASH therefore consolidates these manifestations into one Node-level root cause: adopting a financial value without source evidence matching it to the task requirements. The Edge and Graph records retain the locations where the claim propagated and was used.

Table 16: Level distribution of accepted MASH revisions across the five Finance-Agent folds. Hook-level shares are computed over 64 hook modifications; candidate-level shares are computed over 29 accepted revisions.

Hook-level distribution		
Level	Hook modifications	Share
Node	24	37.5%
Edge	15	23.4%
Graph	25	39.1%
Candidate-level distribution		
Scope	Accepted revisions	Share
Node only	9	31.0%
Edge only	3	10.3%
Graph only	5	17.2%
Cross-level	12	41.4%

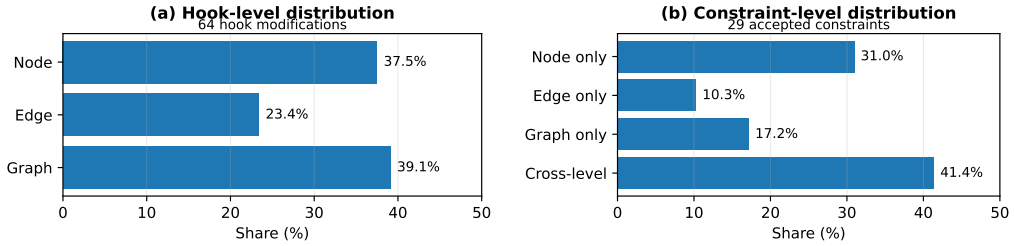


Figure 5: Level distribution of accepted MASH revisions across the five Finance-Agent folds. (a) Hook-level shares over 64 hook modifications. (b) Candidate-level shares over 29 accepted revisions; cross-level revisions modify hooks at two or more levels. Full counts are in Table 16.

We illustrate the subsequent optimization round with three independent candidates derived from the same parent harness, H_0. Candidate A adds local source checks but retains the parent's handoff and stopping rules, potentially ending with unresolved guidance. Candidate B requires the document's publication year to match the target year and adds evidence coverage reporting and follow-up retrieval; this restriction can exclude valid annual reports filed in the following year. Candidate C instead matches the period actually covered or forecast by the source, preserves claim–evidence–source associations during handoffs, and requests evidence for unresolved dependencies.

In this illustrative branch, all candidates pass Stage 1 validity and proposal-consistency checks without MAS execution. Stage 2 requires a mean gain of at least 0.5 points over H_0 on four fixed questions and rejects A. Stage 3 requires at least 0.25 points on eight additional validation questions and rejects B, retaining C for the next round. Both thresholds use a 0–100 score scale.

The illustrated repair under C returns the unresolved guidance to the responsible worker for additional retrieval, updates the evidence-bearing handoff, and completes the comparison. Micron's Q2 announcement provides Q3 FY2024 GAAP gross margin guidance of $25.5\% \pm 1.5$ percentage points. Compared with the actual margin of 26.9%, this yields a beat of $(26.9 - 25.5) \times 100 = 140$ basis points relative to the guidance midpoint. The learned constraint thus matches the period forecast by the evidence, while requiring verified claim–evidence–source binding before aggregation.

C.7 Hyper-parameter sensitivity

We evaluate full MASH on Finance-Agent under the centralized workflow using five-fold evaluation. We vary the search-round limit T and the initial-candidate cap per round, with defaults of $T = 20$ and a cap of three. For the round sweep, we evaluate frozen end-of-round harnesses with the initial-candidate cap fixed at three. For the candidate sweep, we fix $T = 20$ and vary the cap. Initial generation and revisions share a call allowance, so the actual number of initial candidates may be below the cap.

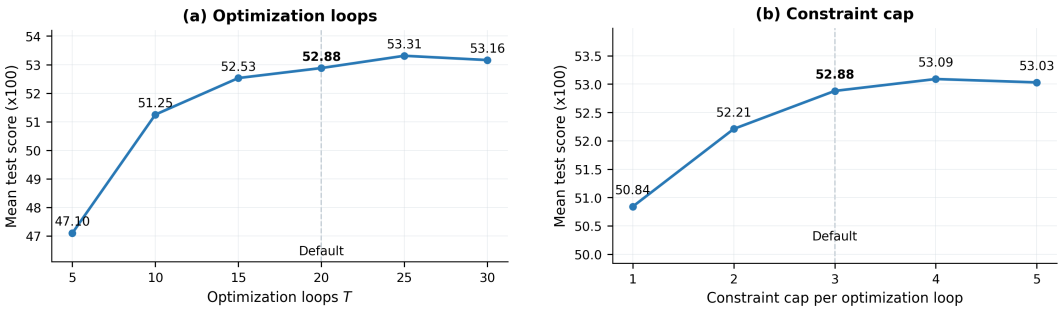


Figure 6: Hyper-parameter sensitivity of full MASH on Finance-Agent under the centralized workflow using five-fold evaluation. (a) $T \in \{5, 10, 15, 20, 25, 30\}$ with an initial-candidate cap of three. (b) Initial-candidate caps of $\{1, 2, 3, 4, 5\}$ with $T = 20$. Scores are multiplied by 100; defaults are marked in both panels.

Search rounds. Figure 6(a) shows rapid early improvement followed by diminishing gains. Extending search beyond the default yields only small additional improvements, and the largest round limit does not achieve the highest observed score.

Initial-candidate cap. Figure 6(b) shows that increasing the cap initially improves performance, with limited gains beyond the default. A cap of three captures most of the observed benefit under the shared call allowance; larger caps do not consistently yield higher scores.

C.8 Search trajectories and resource use

Search trajectories. We examine the F/U/O/R/L organization runs on Finance-Agent under the centralized workflow, starting from the same H_0 and using the common 20-loop schedule. Loop 1 initializes H_0, collects diagnosis trajectories, and generates and filters the first candidate batch; the earliest parent adoption can therefore occur in loop 2. Figure 7 reports the logged five-fold mean-score checkpoints along these searches. U improves more rapidly early in the search, whereas F catches up later and achieves the highest final mean score. L finishes below U but above O and R. A flat segment indicates no change in the plotted mean at those checkpoints; it does not by itself imply that no candidate passed filtering, because a filtered candidate may still fail to produce a positive gain during normal evaluation.

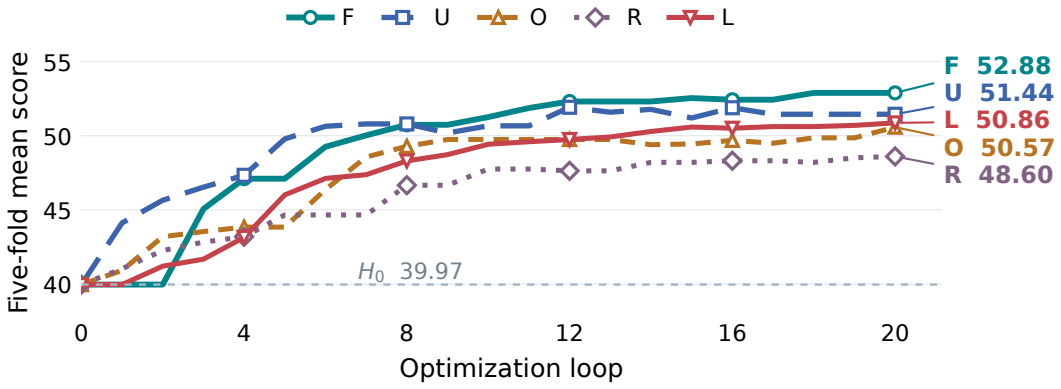


Figure 7: Search trajectories of F/U/O/R/L on Finance-Agent under the centralized workflow. Scores are five-fold means on the 0–100 scale; the horizontal reference line marks $H_0 = 39.97$. Variant definitions follow Table 3. The horizontal axis shows logged search checkpoints rather than matched realized compute. Under the 20-loop protocol, loop 1 initializes H_0 and the earliest parent adoption can occur in loop 2.

Table 18: Comparison of complete failure diagnosis strategies on Finance-Agent under the centralized workflow. F, R, and L denote MASH, free-form analysis and revision, and three-level analysis and revision, respectively. Test scores are on a 100-point scale and averaged over three test repetitions under five-fold evaluation. Costs are averaged per optimization loop.

Metric	MASH (F)	R	L
Test score	**52.88**	48.60	50.86
MAS execution starts / loop	166.3	145.4	159.7
Optimizer tokens / round (k)	90.3	88.9	99.9

Resource use. We use the F/R/L runs from the organization study described in Appendix B.1.2. The strategies match search schedules, not realized MAS execution starts. MAS execution starts and optimizer tokens are averaged over all 100 optimization loops across the five folds, including rounds without accepted updates and excluding initialization and final testing.

Table 18 shows that L achieves a higher final score than R, while MASH achieves the highest score. The F–L comparison combines differences in evidence analysis, revision organization, and implementation requirements, rather than isolating input-side dependency analysis. Compared with L, F uses a similar number of MAS execution starts per loop (166.3 versus 159.7) but fewer optimizer tokens. R has the lowest per-loop usage on both reported resource measures, but also the lowest final score.

D Additional Discussion on Related Work

D.1 Additional Discussion of Agent Harness Optimization

Prompt, context, memory, and tool optimization. GEPA reflectively updates prompts using trajectories and evaluation feedback (Agrawal et al., 2026), while ACE accumulates reusable strategies through incremental context updates (Zhang et al., 2025d). ACON optimizes context-compression guidelines from failure evidence (Kang et al., 2026), and MemEvolve modifies the code implementing memory operations (Zhang et al., 2025b). RewardHarness evolves external tools and skills for a frozen image-preference evaluator (Zhang et al., 2026c).

Executable harness optimization. Meta-Harness allows a proposer to inspect historical code and traces (Lee et al., 2026b). AHE organizes execution evidence and predicted effects of edits for subsequent verification (Lin et al., 2026), while Life-Harness adapts interventions across the agent–environment lifecycle (Xu et al., 2026b). HarnessX and Self-Harness connect failure evidence to candidate modifications and acceptance checks (Chen et al., 2026b; Zhang et al., 2026b). AutoSaddler similarly uses execution-trace diagnosis to produce targeted harness updates (Park et al., 2026).

Evaluation efficiency and continual adaptation. HarnessLens selects informative validation tasks, HarnessBank uses gated verification, and ModularRSI applies successive program, edit, and execution checks (Xu et al., 2026a; Luo et al., 2026; Wu et al., 2026). EvoTest, Continual Harness, and Evo-Harness extend adaptation across repeated episodes, interactions, or tasks (He et al., 2026; Karten et al., 2026; Wei et al., 2026). These settings differ in when feedback is available and whether adaptation continues during evaluation, motivating explicit feedback and inference budgets and independent held-out assessment (Wang et al., 2026a; Ursekar et al., 2026b).