MAS-ProVe: Understanding the Process Verification of Multi-Agent Systems

Vishal Venkataramani [1] **Haizhou Shi** [*1 2] **Zixuan Ke** [†2] **Austin Xu** [2] **Xiaoxiao He** [1] **Yingbo Zhou** [2]
Semih Yavuz [2] **Hao Wang** [‡1] **Shafiq Joty** [‡2]

Abstract

Multi-Agent Systems (MAS) built on Large Language Models (LLMs) often exhibit high variance in their reasoning trajectories. Process verification, which evaluates intermediate steps in trajectories, has shown promise in general reasoning settings, and has been suggested as a potential tool for guiding coordination of MAS; however, its actual effectiveness in MAS remains unclear. To fill this gap, we present MAS-ProVe, a systematic empirical study of process verification for MAS. Our study spans *three verification paradigms* (LLM-as-a-Judge, reward models, and process reward models), evaluated across *two levels of verification granularity* (agent-level and iteration-level). We further examine *five representative verifiers* and *four context management strategies,* and conduct experiments over *six diverse MAS frameworks* on multiple reasoning benchmarks. We find that process-level verification does not consistently improve performance and frequently exhibits high variance, highlighting the difficulty of reliably evaluating partial multi-agent trajectories. Among the methods studied, LLM-as-a-Judge generally outperforms reward-based approaches, with trained judges surpassing general-purpose LLMs. We further observe a small performance gap between LLMs acting as judges and as single agents, and identify a context-length-performance trade-off in verification. Overall, our results suggest that effective and robust process verification for MAS remains an open challenge, requiring further advances beyond current paradigms. Code is available at https://github.com/Wang-ML-Lab/MAS-ProVe.

[*]Work done during an internship at Salesforce AI Research. [†]Project Lead. [‡]Equal Advising. [1]Rutgers University [2]Salesforce AI Research. Correspondence to: Vishal Venkataramani <vishal.venkataramani@rutgers.edu>, Zixuan Ke <zixuan.ke@salesforce.com>, Hao Wang <hw488@cs.rutgers.edu>, Shafiq Joty <sjoty@salesforce.com>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

1. Introduction

Recent advances in Large Language Models (LLMs) have enabled the solution of a wide range of tasks requiring careful high-level planning and scheduling (Ke et al., 2025a), complex logical and mathematical reasoning (Wang et al., 2025), and self-reflection for iterative outcome improvement (Shinn et al., 2023). As these intrinsic capabilities continue to mature, and as the scaling gains of single foundation models begin to plateau, there has been growing interest in constructing coordinated ensembles of LLMs that can collectively understand, decompose, execute, summarize, and reflect on complex tasks (Kim et al., 2025). This emerging paradigm of Multi-Agent Systems (MAS) offers several advantages over Single-Agent Systems (SAS), including reduced context interference and improved parallelism (LangChain, 2025; Ke et al., 2026).

Any potential benefits of MAS are accompanied by increased complexity, increasing the number of potential failure points: final outcome correctness in MAS now depends inherently on the intermediate outputs of a variety of coordinated sub-agents. To weed out mistakes before they influence downstream correctness, many MAS frameworks (Shinn et al., 2023; Du et al., 2024; Liang et al., 2023) explicitly incorporate automatic verification sub-agents to check or refine intermediate outputs, while others work incorporate verifiers for further do test-time scaling of MAS (Brown et al., 2024; Jin et al., 2025). Despite the explicit inclusion of verification modules, subpar evaluation remains a bottleneck in the efficacy of MAS (Cemri et al., 2025). This work attempts to understand the fundamental role of automatic evaluation in MAS by systematically studying evaluators, MAS frameworks, and domains, guided by a central research question:

Do multi-agent systems actually benefit from automatic process-level verification?

In this paper, we address this question along four complementary dimensions: (i) verification type (*generative* versus *scoring-based*), (ii) verification granularity in MAS (verifying each *agentic call* versus each *full iteration*), (iii) context management strategies, and (iv) solvability. Our empirical study spans six representative MAS frameworks, two rea-

soning domains (mathematical problem solving and agentic search), and five distinct evaluators, enabling a systematic and controlled examination of when, how, and to what extent process-level verification benefits multi-agent systems. Through this analysis, we identify both consistent patterns and high-variance behaviors across settings, clarifying the practical limits and trade-offs of applying process verification to MAS.

To facilitate reproducible research and establish a standardized testbed for the community, we introduce a universally applicable protocol, **MAS-Pro**cess **Ve**rification (**MAS-ProVe**), for integrating process-level verification into MAS test-time scaling, as demonstrated in Fig. 1. MAS-ProVe is modular and extensible, operating as a plug-and-play wrapper for *any* off-the-shelf MAS framework and *any* verification method, and requiring only *minimal* code modifications.

Main Observations. We present the first systematic study of the strengths and limitations of process verification in MAS, spanning multiple dimensions including verification types, granularity, context management, and problem solvability.

- **Verification type**: Process verification in MAS does not consistently yield performance improvements across verification types and often exhibits high variance, with judge-based verifiers dominating the reward-based verifiers. This highlights the intrinsic difficulty of reliably evaluating partial multi-agent trajectories.
- **Verification Granularity**: While no single verification granularity universally dominates across MAS frameworks, individual MAS often exhibit a clear preference between agent-level and iteration-level verification.
- **Context Management Strategies**: Comparable performance between only step responses vs context + step(raw) responses, with summarization of responses leading the performance gains in long-context MAS.
- **Solvability**: A question-wise analysis on mathematical reasoning benchmarks shows that the process verification with optimized settings can improve stability, but rarely recovers instances that are fundamentally unsolvable by MAS.

2. Background

In this section, we establish the formal framework that governs our experimental design. Rather than focusing on specific models or datasets, we define the general abstractions for the two core components of our study: the taxonomy of Multi-Agent Systems (Sec. 2.1) and the classes of Automatic Verification (Sec. 2.2). Finally, in Sec. 2.3, we unify these components to define the specific design space of our analysis—granularity, context, and search strategy, which serves as the theoretical basis for the concrete instantiations detailed in the experimental setup.

2.1. MAS Overview

A Multi-Agent System (MAS) consists of multiple LLM-based agents, each equipped with its own context and (sub-)goals, coordinated as a system to perform a given task. We denote the resulting system-level behavior to an input query a *reasoning trajectory*, where the output of each sub-agent can be considered a "step" towards the final output. While a general concept, a variety of recent works (Liu et al., 2024; Chen et al., 2024; Zhang et al., 2025b) have proposed different instantiations of MAS. Broadly, we consider two classes of MAS: (i) *fixed architecture* systems, where the topology of agents and communication edges remains fixed for all problems, and (ii) *adaptive architecture* systems, where the MAS topology may change based on a per-query basis. MAS belonging to the former may either be human designed MAS topologies (e.g., LLM-debate) or instantiated by iteratively searching for the best MAS structure on a small validation set (i.e., "training"). In the latter, MAS tend to self-evolve over multiple iterations at test-time, typically using self-reflection to update agent roles and communication edges.

2.2. Automatic Verification

To meet demands for scalable verification, automatic verifiers have been deployed to provide feedback signals about output and intermediate state correctness in reasoning settings (Snell et al., 2025; Zhou et al., 2025b; Shao et al., 2025). There are three primary classes of verifiers: (i) *reward model (RMs)*, which produce a continuous score for an input and terminal output state, (ii) *process reward models (PRMs)*, which produce continuous scores for intermediate states, measuring likelihood of reaching a correct outcome from a given state, and (iii) *generative verifiers (LLM-as-a-Judge)*, which outputs feedback and evaluation in natural language. For LLM-as-a-Judge baselines, we analyze both general purpose LLMs as judges and specialized, state-of-the-art finetuned verifier models.

2.3. Marrying MAS and Verification

Integrating verification into MAS settings gives rise to a search-based optimization problem that we seek to analyze, i.e., yielding multiple execution/trajectories in parallel and then letting the verifier to pick the most promising candidate. We choose three crucial facets for analysis:

Verification Types. To broadly cover the dominant classes of automatic verifiers used in prior work, MAS-ProVe includes all three verifier types in our analysis. To the best of our knowledge, existing verifiers are not explicitly trained on

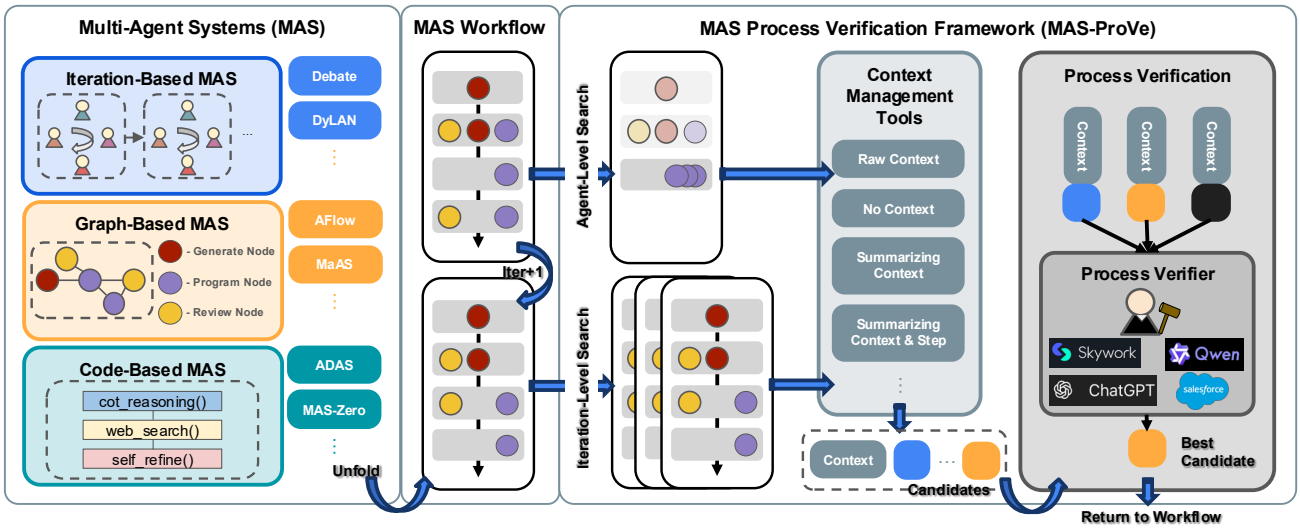

Figure 1. **Overview of Multi-Agent System Process Verification Framework (MAS-ProVe).** The execution of *any* multi-agent system (MAS) can be abstracted into a unified workflow, where stages are executed sequentially and atomic LLM calls within each stage may run in parallel. MAS-ProVe systematically evaluates the integration of MAS with process verification by performing parallel search at two complementary granularities: *agent-level* and *iteration-level*, while remaining agnostic to the choice of process verifier. Prior to verification, customizable *context-management strategies* can be applied to partial multi-agent trajectories, enabling flexible control over the information passed to the verifier and the selection of the best candidate to return to the workflow.

multi-agent trajectories; as a result, their application to MAS constitutes a strictly out-of-distribution setting. Among these approaches, LLM-as-a-Judge is typically instantiated using general-purpose LLMs with strong generalization capabilities. In contrast, reward models—particularly process reward models—are commonly trained on single-agent, verifiable reasoning trajectories (e.g., mathematical problem solving). Consequently, we hypothesize that LLM-as-a-Judge is more naturally suited to MAS test-time scaling than reward-based alternatives.

Verification Granularity. Verification-based interventions can occur at different levels of granularity. We study two concrete levels, as shown in Fig. 1 (middle): the *Agent-Level*, where the outputs of each sub-agent are explicitly verified, and the *Iteration-Level*, where the MAS itself is evaluated as a whole. The former enables catching fine-grained errors before they corrupt other sub-agent behavior, but introduces significant overhead. The latter takes a more global perspective potentially at the cost of finding step-level errors.

Context Management. The reliability of process verification critically depends on how the reasoning state is represented to the verifier. In MAS, trajectories may span many interaction turns; naively providing the full raw history can introduce bias due to context saturation, while conditioning only on the current step ignores the state dependencies required to assess reasoning continuity. To empirically identify a "sweet spot" in contextual conditioning, we evaluate four context specifications with increasing information density: *(i) current-step only*, *(ii) summarize(context) + current step*, *(iii) summarize(context + current step)*, and *(iv) full context + current step*.

Solvability. To disentangle the effect of process verification from the intrinsic reasoning capability of the underlying agents, we analyze MAS performance through the lens of *problem solvability*. A verification signal can guide the system toward a correct solution only if such an outcome lies within the agent's generation horizon. Consequently, it is essential to first characterize the difficulty of the benchmarks on which the MAS operates. We partition each benchmark into three difficulty levels, *Easy*, *Medium*, and *Hard* based on the empirical pass rate of the unguided CoT(Chain-of-Thought) baseline, and introduce task-specific metrics to quantify how process verification interacts with solvability. Specifically, we consider: *(i) Performance Gain*, *(ii) Evaluator Stability*, and *(iii) Intra-MAS Resurrection*.

3. Experimental Setup

Benchmarks. Our analysis focuses on four representative reasoning domains: mathematics (**AIME24** and **AIME25**, (AIME, 2025)), agentic assistance (**GAIA** (Mialon et al., 2023)), coding (**HumanEval** (Chen et al., 2021)), and scientific reasoning (**GPQA** (Rein et al., 2023)). To support MAS baselines that require training or tuning, we partition each benchmark into a validation set (20%) and a test set (80%), and reuse the same splits across all methods for fair comparison.

MAS Baselines and Implementation Details. We conduct a comprehensive evaluation on six state-of-the-art MAS frameworks with both *fixed architectures*, including LLM-Debate (**Debate**) (Du et al., 2024), Agentic Workflow (**AFlow**) (Zhang et al., 2025b), and Automated Design of Agentic Systems (**ADAS**) (Hu et al., 2025), and *adaptive architectures*, including Dynamic LLM-Powered Agent Network (**DyLAN**) (Liu et al., 2024), Multi-agent Architecture Search (**MaAS**) (Zhang et al., 2025a), and MAS with Zero Supervision (**MAS-Zero**) (Ke et al., 2025b).

To balance cost efficiency and reasoning capability, we adopt **GPT-5-Mini** as the primary backbone model across all baselines. Only for HumanEval, we use the **GPT-4.1-nano** model; this prevents the performance ceiling that stronger models would hit on this simpler task, allowing us to accurately observe the delta introduced by process verification. For the agentic benchmark GAIA, all MAS frameworks are further equipped with a DuckDuckGo search tool via OpenAI's tool-calling interface. This enables dynamic external information retrieval and facilitates a systematic evaluation of process verification granularity in tool-assisted reasoning settings. Detailed configurations of the MAS baselines can be found in Appendix A.2.

Verifiers. We consider three main verifiers, one for each category: **Skywork-Reward-V2-Llama-3.1-8B** (Liu et al., 2025a) as an RM, **Qwen2.5-Math-PRM-7B** (Zhang et al., 2025d) as a PRM, and **GPT-5-Mini** as an LLM-as-Judge. In extended analysis, we also consider the weaker GPT-4o-Mini and the finetuned verifier **FARE-20B** (Xu et al., 2025), which is trained from **GPT-OSS-20B** (Agarwal et al., 2025).

Integrating Verifiers into MAS via Search. Given verification scores produced by the chosen evaluators, the system must determine how to advance the reasoning process. We adopt a *greedy best-first search* strategy at the specified verification granularity for every MAS framework to ensure a scalable, effective, and fair comparison. At each state, the MAS generates N=3 candidate continuations and selects the highest-scoring/ranking candidate according to the verifier to define the next state. This procedure effectively transforms the MAS from a purely stochastic generator into a verifier-guided search process. To further enrich the search heuristics and provide a more comprehensive exploration of this bounded space, we additionally implement Beam Search (beam width=3) on top of the Debate framework using our optimal Generative Judge, as detailed in Appendix B.4.

`MAS-ProVe` Framework. A broad overview of our framework is illustrated in Fig. 1. *Conceptually,* the execution of *any* multi-agent system can be abstracted into a unified workflow, where stages are executed sequentially and atomic LLM calls within each stage may run in parallel. `MAS-ProVe` systematically evaluates the integration of MAS with process verification by performing parallel search at two complementary granularities: *agent-level* and *iteration-level*, while remaining agnostic to the choice of process verifier. Prior to verification, customizable *context-management strategies* can be applied to partial multi-agent trajectories, enabling flexible control over the information passed to the verifier and the selection of the best candidate to return to the workflow.

Practically, the `MAS-ProVe` framework is implemented as an isolated *client-middleware-server* architecture:

- **Client side.** The client executes the original MAS agentic logic *without modification,* as at each reasoning step, the wrapper function automatically executes the original LLM call for N times and interfaces with the middleware to obtain the selected continuation and resumes execution along the chosen path.
- **Middleware side.** A context-management tool intercepts the agent's execution, retrieves the accumulated context from the MAS, and re-organizes the context. Then these partial trajectories are forwarded to the server for evaluation.
- **Server side.** The server hosts a pool of verifiers that score and rank the received partial trajectories. The resulting rankings are returned to the client via the middleware, which selects the highest-ranked continuation and enables the MAS to proceed.

Remark on Computational Overhead. `MAS-ProVe`'s systematic study inherently incurs additional computational overhead and is not intended to reflect the practical integration of process verification into MAS. A detailed breakdown of the wall-clock time is provided in Appendix B.1.

4. Main Results

This section expands on the empirical insights derived from experiments conducted along the four dimensions introduced in Sec. 2. Sec. 4.1 examines different *verification types*, comparing generative and scoring-based verifiers and analyzing the impact of judge specialization. Sec. 4.2 studies *verification granularity*, characterizing each MAS's preference between agent-level and iteration-level process verification for maximizing performance gains. Sec. 4.3 investigates *context management strategies*, quantifying the trade-off between information retention and verification reliability. Finally, Sec. 4.4 decomposes performance by *problem solvability*, isolating the role of process verification in recovering previously unsolvable queries versus improving stability across varying difficulty levels.

4.1. Verification Paradigms

We analyze the extent of judge verification through different choices of verifiers. The experiments on Table 1 and 2

Table 1. **Main Study of MAS-ProVe on Verification Types and Granularity.** We investigate two core dimensions of process verification for multi-agent systems: *(i)* the choice of process verifier (LLM-as-a-Judge, Reward Model, or Process Reward Model), and *(ii)* the granularity at which verification is applied. All experiments are repeated three times, and we report the mean and standard deviation. Values reported under the *No ProVe* row for each MAS correspond to the *baseline performance.* Higher values indicate better performance throughout. **Boldface** denotes the best result in each setting. Green cells indicate improvements over the baseline, while red cells indicate degradations relative to the baseline.

Method	MAS-Arch	ProVe Level	AIME24			AIME25			GAIA		
			Judge	RM	PRM	Judge	RM	PRM	Judge	RM	PRM
Debate	Fixed	No ProVe		74.13±3.44			61.87±3.21			31.07±2.07	
		Agent	**81.11±1.92**	71.67±1.44	68.10±4.50	**62.97±3.38**	54.45±3.85	57.12±8.85	**30.28±1.51**	25.05±2.33	26.20±0.14
		Iteration	72.67±5.97	**81.11±3.85**	70.34±3.07	53.33±8.82	**57.78±7.70**	54.59±2.18	**32.68±3.11**	20.60±1.21	23.20±4.50
AFlow	Fixed	No ProVe		65.67±5.09			59.72±6.37			11.50±3.25	
		Agent	**76.39±6.36**	76.22±6.59	74.17±3.01	**58.33±4.17**	58.32±11.04	56.94±8.68	**21.68±2.08**	18.08±5.52	14.62±2.03
		Iteration	**69.44±8.67**	52.73±10.52	61.11±2.41	**60.39±6.26**	54.16±5.89	49.99±4.18	18.62±0.78	**21.64±1.64**	13.85±2.55
ADAS	Fixed	No ProVe		62.49±4.19			54.17±4.15			19.30±0.92	
		Agent	73.33±12.83	**73.93±8.26**	68.07±9.64	58.33±0.00	**61.10±2.42**	48.90±2.42	20.90±2.50	**22.10±2.50**	19.70±4.33
		Iteration	**70.83±15.03**	66.67±11.06	63.87±6.36	**61.10±4.85**	59.73±4.79	54.23±0.00	**22.90±2.08**	19.70±1.83	18.53±3.01
MaAS	Adaptive	No ProVe		76.39±6.36			58.33±7.22			20.97±4.90	
		Agent	80.55±8.67	**84.03±6.73**	83.33±4.17	**61.09±4.83**	52.08±2.94	59.70±2.42	**26.89±5.11**	25.36±6.41	24.70±0.86
		Iteration	77.22±10.48	**86.11±6.37**	81.94±6.36	**68.06±6.36**	63.89±2.41	63.88±8.68	**31.21±3.60**	27.29±2.55	24.26±0.97
DyLAN	Adaptive	No ProVe		81.11±5.09			65.55±5.09			19.37±0.95	
		Agent	**85.56±1.93**	76.67±0.00	77.78±3.85	70.00±3.85	**72.22±5.09**	67.78±3.85	18.12±2.02	**19.41±1.68**	16.50±1.37
		Iteration	**84.44±1.93**	78.89±1.92	82.22±10.71	71.11±3.85	**73.33±5.78**	67.77±8.39	**15.50±1.41**	14.83±1.53	9.70±1.37
MAS-Zero	Adaptive	No ProVe		37.78±1.92			35.56±1.93			10.47±0.68	
		Agent	**47.78±1.92**	38.89±1.92	33.34±5.77	**42.22±3.85**	28.87±5.10	20.00±3.85	17.36±0.97	**19.14±1.06**	10.14±0.62
		Iteration	**50.00±3.33**	45.00±2.36	37.78±1.92	**27.78±1.92**	25.56±5.09	25.56±8.39	**17.95±1.11**	14.94±0.62	8.20±0.71

are conducted by fixing the context to *Raw/Full History* to establish a baseline.

Intrinsic Difficulty of Partial Trajectory Verification. As evidenced by the high variance in the *Debate*, *AFlow*, and *MAS-Zero* rows of Table 1 and 2, process verification does not consistently yield monotonic performance improvements. Although other methods show a natural sign of performance gains compared to the baseline, the pattern is noisy which highlights the challenge of evaluating the partial trajectories of MAS.

Generative Judges Outperform Scoring-based Reward Models. Despite the high variance, a clear hierarchy is noticed when comparing the *LLM-as-a-Judge* column against the *Reward Model* (Skywork-Reward-V2) and *Process Reward Model* (Qwen2.5-Math-PRM) columns. Generative judges achieved higher accuracy in 24 out of 36 experimental configurations across Agent-Call and Iteration levels. We attribute this performance gap to the nature of the training data for the reward model baselines. While the scalar baselines are highly specialized for specific domains (e.g., math steps or final outcomes), general-purpose generative judges appear better equipped to interpret the heterogeneous, out-of-distribution dynamics of MAS trajectories. This supports our hypothesis that the flexible, natural language reasoning of general LLMs offers a stronger steering signal for multi-agent systems than the rigid scalar values of specialized reward models.

Diminishing Returns on Judge Scaling. In Table 3, here, we fix the process verification level to *Agent Call* (for granular viewing) and isolate the impact of the verifier's model architecture. The performance gap between using a smaller generalist judge (GPT-4o-mini) and a reasoning judge (GPT-5-mini) is significantly smaller than the gap observed when these models act as solvers. This indicates that the reasoning threshold required to *verify* a multi-agent trajectory is lower than the threshold required to *generate* it, suggesting that cost-effective, smaller models can serve as effective supervisors for larger systems.

Understanding Specialized Verification Models. Notably, the task-specific evaluator **FARE-20B** generally outperforms both general-purpose models (GPT-5-mini and GPT-4o-mini). For example, FARE-20B leads to the highest performance for *MAS-Zero* on AIME24 (51.11±1.92) and GAIA (21.15±2.72), *ADAS* on AIME25 (66.66±4.15) and GAIA (22.50±4.85) across all our experiments. These results suggest that evaluation-specific finetuning can yield additional gains over general-purpose LLMs-as-Judges.

4.2. Verification Granularity

Table 1 shows that **no single verification granularity universally dominates across multi-agent systems**. Neither agent-level nor iteration-level verification consistently outperforms the other across MAS architectures, benchmarks, and verifier types, indicating that granularity choice is highly architecture-dependent rather than a global heuristic.

Nevertheless, **individual MAS often exhibit clear pref-**

Table 2. **Main Study of MAS-ProVe on Verification Types and Granularity (on Coding and Scientific Reasoning).** Higher values indicate better performance throughout. **Boldface** denotes the best result in each setting. Green cells indicate improvements over the baseline, while red cells indicate degradations relative to the baseline.

Method	MAS-Arch	ProVe Level	HumanEval			GPQA		
			Judge	RM	PRM	Judge	RM	PRM
Debate	Fixed	No ProVe		76.40±1.56			59.13±3.50	
		Agent	**78.35±0.49**	76.10±0.17	78.04±0.00	**66.82±3.36**	52.48±2.79	55.25±0.28
		Iteration	**77.72±0.45**	76.20±0.85	77.02±0.71	**70.95±0.35**	69.19±0.71	58.53±5.78
AFlow	Fixed	No ProVe		88.78±1.17			46.33±0.96	
		Agent	**89.80±1.19**	88.14±0.51	73.66±4.86	**54.93±2.65**	44.24±1.32	39.72±1.78
		Iteration	92.58±0.48	**92.75±0.54**	89.28±1.11	**52.20±3.55**	40.56±2.22	37.73±1.78
ADAS	Fixed	No ProVe		95.42±3.12			66.54±1.83	
		Agent	94.82±2.41	**95.03±3.83**	94.08±4.37	68.94±3.64	**69.16±4.31**	60.79±4.82
		Iteration	**96.38±2.63**	95.20±2.88	96.20±1.65	**69.54±2.90**	55.23±3.76	51.65±2.31
MaAS	Adaptive	No ProVe		96.18±1.08			66.23±1.44	
		Agent	95.93±0.44	95.80±2.70	**96.18±0.00**	**67.90±0.86**	66.66±0.89	65.09±1.33
		Iteration	**97.33±0.54**	96.56±0.54	96.18±2.16	**71.06±2.67**	68.33±1.46	61.62±0.87
DyLAN	Adaptive	No ProVe		90.23±0.00			68.18±1.43	
		Agent	**93.65±0.35**	91.16±1.29	89.72±0.74	**78.76±2.91**	76.46±0.29	67.92±2.50
		Iteration	**94.20±0.42**	92.38±0.43	93.62±0.40	**72.20±1.41**	69.44±1.77	68.68±1.43
MAS-Zero	Adaptive	No ProVe		53.63±3.00			60.72±3.50	
		Agent	**55.75±0.00**	53.63±1.62	50.91±2.57	**68.00±1.24**	61.41±1.13	55.99±0.41
		Iteration	**54.23±2.14**	51.81±1.29	52.72±0.00	61.31±2.86	**64.06±0.36**	53.77±1.48

Table 3. **Study of MAS-ProVe on LLM-as-a-Judge as Verifiers. Boldface** denotes the best performance.

Method	AIME24			AIME25			GAIA		
	GPT-4o-mini	GPT-5-mini	FARE-20B	GPT-4o-mini	GPT-5-mini	FARE-20B	GPT-4o-mini	GPT-5-mini	FARE-20B
Debate	72.23±3.87	**81.11±1.92**	76.67±1.65	54.47±6.32	62.97±3.38	**64.43±5.38**	26.85±4.42	**30.28±1.51**	27.87±2.66
AFlow	76.39±8.67	76.39±6.36	**81.94±4.82**	58.33±4.17	58.33±4.17	**61.10±2.42**	19.27±4.17	**21.68±2.08**	16.55±0.85
ADAS	69.45±4.81	**73.33±12.83**	68.07±9.64	63.87±6.36	58.33±0.00	**66.66±4.15**	20.50±2.08	20.90±2.50	**22.50±4.85**
MAAS	83.31±8.30	80.55±8.67	**84.73±6.36**	**62.50±4.17**	61.09±4.83	58.33±4.17	26.50±3.19	26.89±5.11	**28.31±0.85**
DyLAN	83.33±3.34	85.56±1.93	**86.67±3.34**	**73.33±5.78**	70.00±3.85	**73.33±3.34**	**19.37±0.95**	18.12±2.02	18.37±0.98
MAS-Zero	42.22±5.09	47.78±1.92	**51.11±1.92**	30.00±3.33	**42.22±3.85**	38.89±1.92	16.62±1.48	17.36±0.97	**21.15±2.72**

erences. *Debate* and *MAS-Zero* benefit more consistently from *iteration-level* verification—especially with LLM-as-a-Judge—while *AFlow* and *ADAS* show stronger gains under *agent-level* verification, suggesting better alignment with finer-grained control. *MaAS* improves under both granularities but with benchmark-specific differences, whereas adaptive architectures such as *DyLAN* display mixed behavior, favoring iteration-level verification for mathematical reasoning and agent-level verification for GAIA.

Overall, these results highlight that verification granularity interacts closely with MAS coordination dynamics: although no granularity is universally optimal, **aligning granularity with the MAS architecture is crucial for realizing the benefits of process-level verification.**

4.3. Context Management Strategies

In Table 4, we investigate the impact of information fidelity on the judge decision-making process. The process verification level is again fixed to *Agent Call*.

Context-Length vs Performance Trade-Off. Results in Table 4 demonstrate that process verification with minimal context (Step-Only) performs comparably to, and occasionally better than, evaluation with Full Context. This suggests a strong need to understand the sweet-spot of context length that provides the right amount of context alongside not being vague. While full history contains necessary dependencies, it also introduces noisy tokens that can possibly degrade the judge's attention. Our data confirms a distinct context-length-performance trade-off in MAS process verification, where the utility of additional context diminishes as the noise-to-signal ratio increases.

Summarized Context works best for MAS with long context. The **MaAS** and **DyLAN** baselines produce MAS instances that take up significantly more tokens than other baselines, as shown in Table 5. As a result, when evaluating these frameworks, the input context of the verifier must be carefully curated. In Fig. 2, *Summarized Context* strategies (Square/Triangle markers) consistently dominates *Raw History* (Diamond markers) in both performance and efficiency. For frameworks like MaAS, the Raw strategy incurs a ∼3x token overhead while yielding comparatively lower accuracy. Further analysis/plots with respect to token growth, efficiency are presented in Appendix B.7.

Table 4. **Comparing Different Context Management Strategies for Process Evaluation. Boldface** denotes the best performance.

Method	AIME24				AIME25				GAIA		
	Step-Only	Raw	Summary	Cont. Summary	Step-Only	Raw	Summary	Cont. Summary	Step-Only	Raw	Summary
Debate	**81.11**±1.92	**81.11**±1.92	77.78±3.85	-	62.97±3.38	62.97±3.38	**70.00**±6.67	-	**30.28**±1.51	**30.28**±1.51	24.78±0.47
AFlow	79.16±4.17	76.39±6.36	76.39±6.36	**80.55**±4.81	65.28±6.37	58.33±4.17	**69.44**±2.40	66.67±4.17	18.05±2.11	**21.68**±2.08	16.86±3.18
ADAS	72.23±6.37	73.33±12.83	70.83±11.00	**73.63**±12.01	**66.67**±0.00	58.33±0.00	65.27±4.79	**66.67**±0.00	**22.50**±1.83	20.90±2.50	22.10±3.02
MAAS	81.94±2.41	80.55±8.67	**84.72**±4.82	**84.72**±6.37	65.28±13.39	61.09±4.83	62.49±4.19	**68.06**±2.40	24.10±1.70	**26.89**±5.11	23.45±2.62
DyLAN	**86.67**±3.34	85.56±1.93	82.22±5.09	**86.67**±3.34	**80.00**±6.67	70.00±3.85	**80.00**±0.00	78.89±5.09	**20.67**±1.10	18.12±2.02	14.10±2.56
MAS-Zero	48.88±5.08	47.78±1.92	**48.89**±1.92	42.22±5.09	**45.56**±10.18	42.22±3.85	32.22±5.09	33.33±3.34	17.28±1.39	**17.36**±0.97	15.07±0.54

Table 5. **Cost Efficiency and Trajectory Analysis.** Comparison of average token consumption per question across context strategies, alongside the average number of reasoning steps per trajectory.

	Avg.	Avg. Tokens per Question by Context Strategy			
Architecture	Steps	Step-Only	Raw	Summary	Summ. + Step
DyLAN	4.4	1,196.6	2,280.7	467.1	1,650.2
MaAS	4.8	3,000.6	8,826.4	650.6	3,638.6
AFlow	5.2	365.8	1,754.9	574.1	913.7
ADAS	11.1	312.6	915.1	560.1	832.5
MAS-Zero	21.6	330.8	1,409.7	589.9	861.0

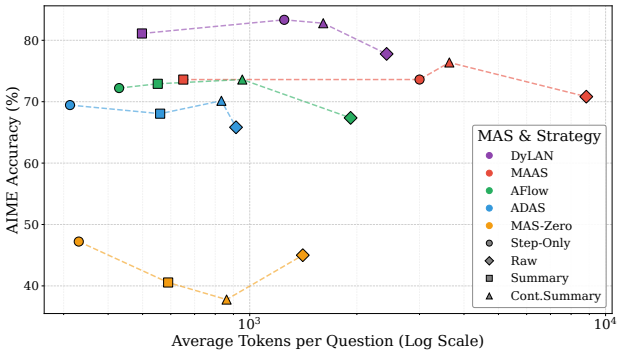

Figure 2. Accuracy vs. Token Cost (Log Scale) across MAS architectures. Strategies closer to the top-left corner are to be preferred. Summarized contexts consistently outperform Raw History while consuming significantly fewer tokens.

Simply changing the judge or the context management can yield large gains. Here, we aggregate our findings from Table 3 and Table 4, showing that even optimizing one of the choice of judge model and the choice of judge context management can yield substantial gains over blindly picking an evaluator. In Fig. 3, we plot the best performance of each MAS framework under different judge choices and context-management strategies on AIME24 and AIME25, denoted as "Best Configuration". We additionally compute **Pass@3** of standard MAS baseline runs. We find that some MAS frameworks outperform Pass@3 when paired with appropriately chosen verifications, such as AFLow on AIME24 or DyLAN on AIME25. Notably, the best judge configuration typically outperforms RM and PRM baselines by significant margins.

In addition to these core dimensions, we investigate the underlying failure modes that contribute to occasional performance degradation. Specifically, we isolate instances of verifier misranking and demonstrate how the generative judge's reliability is highly sensitive to the provided context format. A detailed ablation study exploring this context-driven brittleness is provided in Appendix B.2.

4.4. Analysis of Solvability

We establish a difficulty baseline using a standard Chain-of-Thought (CoT) policy. We execute the base model (GPT-5-mini) on every question in the AIME 24 and AIME 25 datasets for $N=30$ independent trials to account for stochastic variance. The detailed stratification on both the datasets can be viewed in Appendix A.

Building on this difficulty stratification, we proceed with a granular, question-wise comparative analysis to disentangle the impact of process supervision from inherent task complexity.

Methodology. We conduct comparisons between the standard MAS baseline runs, the Agent call search executions guided by the **GPT-5-mini Judge**, and the best result acheiving configuration across our experiments. This formulation allows us to quantify these three critical metrics:

- **Performance Gain:** The absolute delta in pass rate between the baseline and the process-evaluated agents within each difficulty stratum.
- **Evaluator Stability:** The consistency of the judge's decisions, measured by the variance of outcomes (e.g., 0/3, 1/3, 2/3 or 3/3 success) across trials.
- **Intra-MAS Resurrection:** The recovery rate of baseline-failed questions (0/3), measuring whether process verification can salvage completely failed cases into partial or full successes.

By isolating these variables, we can determine whether Process verification acts as a deterministic guide or introduces productive stochasticity. The resulting performance dynamics and stability profiles yields the following observations.

Process verification improves performance primarily by enhancing stability, with marginal utility on hard tasks. As illustrated in Fig. 4, the relative performance lift provided by our framework correlates positively with task difficulty. To pinpoint the source of this improvement, we analyze whether verification *stabilizes* noisy baseline outputs or *res-*

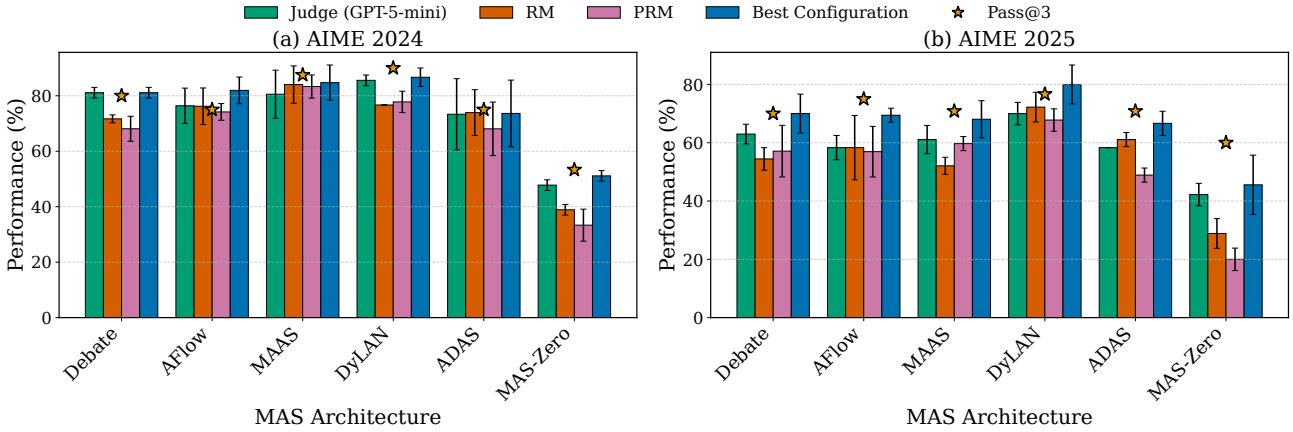

Figure 3. Performance across MAS architectures on Mathematical Reasoning Benchmarks using different process verification methods: Judge (GPT-5-mini), RM (Skywork-Reward-V2-Llama-3.1-8B), PRM (Qwen2.5-Math-PRM-7B), and Best Configuration. Error bars show standard deviation over three runs. Stars indicate Pass@3 performance.

urrects fundamentally unsolvable problems. We observe that absolute gains remain marginal across all strata: *Easy* (70–100% baseline pass rate), *Medium* (30–70%), and *Hard* (0–30%). While ceiling effects likely constrain the Easy and Medium strata, the minimal shift in Hard tasks points to intrinsic solvability bounds within the underlying MAS. Instead, the primary driver of performance appears to be variance reduction; as shown in Fig. 5, optimal configurations yield a marked increase in *Stable Successes* (Green). This suggests that the utility of process verification lies less in solving novel hard problems, and more in systematically stabilizing solvable queries that suffer from generation variance.

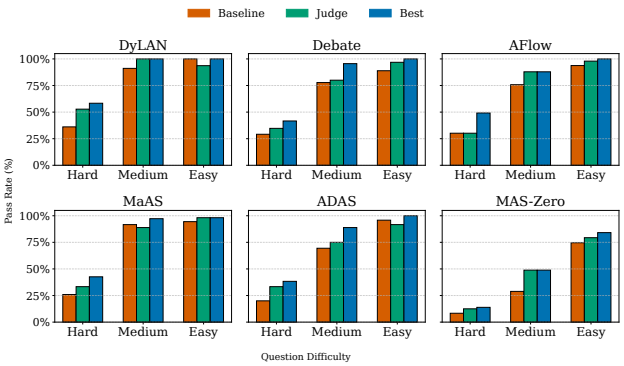

Figure 4. Average Pass rate on Mathematical Reasoning tasks by difficulty stratum. Baseline (red), Process Eval (green), and Best (blue) configurations compared across Hard, Medium, and Easy questions for six MAS architectures.

Process verification encounters a fundamental solvability ceiling. Complementing our findings on stability, analysis suggests that the framework's utility is strictly bounded by the agent's intrinsic generation horizon. As detailed in Fig. 6, for queries classified as "fundamentally unsolvable" (where the baseline policy yields 0% success across all sampled trajectories), process verification rarely triggers a *Full Resurrection* (Green). The majority of these instances result in *Partial Resurrection* (Orange) or remain *Still Failed* (Red). This confirms that while the verifier effectively prunes incorrect reasoning paths, it cannot completely synthesize correct ones that lie outside the agent's underlying reasoning manifold.

To separate problems that are intrinsically unsolvable by the MAS generator from a failure of the verifier's ranking capacity, we conducted an exhaustive search experiment (Pass@128) across AIME questions of varying baseline difficulty. Our findings confirm that **for out-of-distribution queries, the MAS never generates the solution,** a fundamental generation limit that no verifier can overcome. Conversely, when the correct trajectory exists within the search space, the Generative Judge successfully identifies it and guides the search away from suboptimal solution trajectories. A detailed breakdown of this Pass@128 analysis is provided in Appendix B.3.

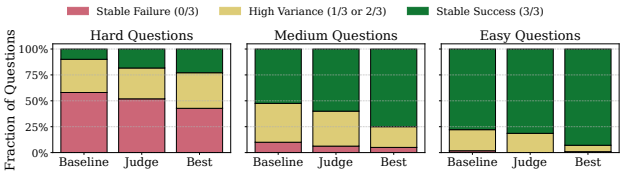

Figure 5. Stability distribution aggregated across all MAS architectures on mathematical reasoning tasks. Stacked bars show trial outcome proportions by difficulty: Stable Failure (red, 0/3), High Variance (yellow, 1-2/3), and Stable Success (green, 3/3) for Baseline, Judge, and Best configurations.

5. Related Work

Inference-time Automatic MAS. Recent advancements in Multi-Agent Systems have shifted towards inference-

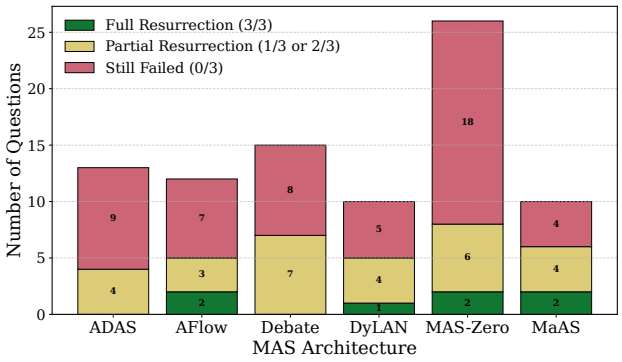

Figure 6. **Intra-MAS Resurrection Analysis.** The chart visualizes the recovery yield of the best process verification configuration from our experiments on these specific failure cases. *Full Resurrection* (Green) demonstrates that process-guided search can rarely navigate to solution states that are unreachable by the baseline.

time optimization, where agent structures or workflows are adapted dynamically to the input query. Frameworks like **MASS** (Zhou et al., 2025a) and **MaAS** (Zhang et al., 2025a) utilize rejection sampling and masking mechanisms to select effective sub-agent configurations on the fly. More aggressive optimization approaches, such as **ADAS** (Hu et al., 2025) and **AFlow** (Zhang et al., 2025b), treat MAS execution as a code generation task, employing search algorithms like MCTS to discover optimal workflows. **MAS-Zero** (Ke et al., 2025b), being a noble outlier, attempts fully inference-time adaptation without external validation sets. With the existing architectures focus being largely on overall architectural optimization, our protocol effectively adds a quality-assurance layer on top of these inference-time frameworks. This complements with the current landscape, by steering the intermediate outputs of the MAS.

Verification for MAS. While early strategies for verification relied on scalar RMs to guide search, recent work suggests that generative verifiers may offer superior guidance for complex reasoning. Zhang et al. (2025c) demonstrate that trained generative verifiers outperform scalar RMs in Best-of-N reranking tasks. Verification serves as the engine for Test-Time Scaling (TTS). Snell et al. (2025) and Liu et al. (2025b) show that compute-optimal scaling strategies rely heavily on the interaction between the policy model and the process verifier. Zhou et al. (2025b) further introduced the JETTS benchmark to systematically evaluate these judges, showing that generative evaluators can match or exceed the performance of specialized reward models in reranking contexts. Our work extends this by investigating how these scaling benefits translate to the granular, intermediate steps of Multi-Agent Systems.

A critical factor in process verification is the alignment between the generator and the verifier. Chen et al. (2025) observe a strong positive correlation between generation capability and evaluation accuracy when the same model is used for both roles, a setup we adopt by using GPT-5-mini as both for the generating agent and judge. However, this setup introduces risks of self-preference bias (Panickssery et al., 2024), where evaluators may favor their own reasoning patterns. Additionally, Krumdick et al. (2025) find that evaluator performance degrades significantly when the task difficulty exceeds the model's latent reasoning capability, a limitation we analyze through our Resurrection results.

6. Conclusion

In this work, we conduct a comprehensive empirical study of process verification in multi-agent systems, systematically examining the interactions among verification type, verification granularity, and context management strategies. Our results show that, although process-level verification can improve stability on queries that are already within the agents' solvable regime, it rarely enables recovery from fundamentally unsolvable states, underscoring the limits of current verification-based steering mechanisms. We further release a modular engineering framework that serves as a plug-and-play wrapper for *any* off-the-shelf MAS and *any* process verifier, enabling reproducible evaluation and future extensions. Overall, our findings suggest that integrating MAS with automated process verifiers is ***not yet*** a universal solution: reliably assessing and ranking partial multi-agent trajectories remains a central open challenge that cannot be resolved by straightforward verification alone.

Impact Statement

This research advances methods for making large language model collaboration more capable, reliable, and interpretable, through a systematic empirical study of how process verification interacts with the existing multi-agent sysmtems. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

Acknowledgement

We thank all reviewers, AC, and SAC for their valuable comments. HW is supported by Amazon Faculty Research Award, Microsoft AI & Society Fellowship, NSF CAREER Award IIS-2340125, NIH grant R01CA297832, and NSF grant IIS-2127918.

References

Agarwal, S., Ahmad, L., Ai, J., Altman, S., Applebaum, A., Arbus, E., Arora, R. K., Bai, Y., Baker, B., Bao, H., et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

AIME. American Invitational Mathematics Examination. Mathematical Association of America, 2025. https://maa.org/maa-invitational-competitions/.

Brown, B., Juravsky, J., Ehrlich, R., Clark, R., Le, Q. V., Ré, C., and Mirhoseini, A. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL https://arxiv.org/abs/2407.21787.

Cemri, M., Pan, M. Z., Yang, S., Agrawal, L. A., Chopra, B., Tiwari, R., Keutzer, K., Parameswaran, A., Klein, D., Ramchandran, K., Zaharia, M., Gonzalez, J. E., and Stoica, I. Why do multi-agent llm systems fail?, 2025. URL https://arxiv.org/abs/2503.13657.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Chen, W., Su, Y., Zuo, J., Yang, C., Yuan, C., Chan, C.-M., Yu, H., Lu, Y., Hung, Y.-H., Qian, C., Qin, Y., Cong, X., Xie, R., Liu, Z., Sun, M., and Zhou, J. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=EHg5GDnyq1.

Chen, W.-L., Wei, Z., Zhu, X., Feng, S., and Meng, Y. Do llm evaluators prefer themselves for a reason?, 2025. URL https://arxiv.org/abs/2504.03846.

Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., and Mordatch, I. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=zj7YuTE4t8.

Hu, S., Lu, C., and Clune, J. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=t9U3LW7JVX.

Jin, C., Peng, H., Zhang, Q., Tang, Y., Metaxas, D. N., and Che, T. Two heads are better than one: Test-time scaling of multi-agent collaborative reasoning. *arXiv preprint arXiv:2504.09772*, 2025.

Ke, Z., Jiao, F., Ming, Y., Nguyen, X.-P., Xu, A., Long, D. X., Li, M., Qin, C., Wang, P., Savarese, S., Xiong, C., and Joty, S. A survey of frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems, 2025a. URL https://arxiv.org/abs/2504.09037.

Ke, Z., Xu, A., Ming, Y., Nguyen, X.-P., Xiong, C., and Joty, S. MAS-ZERO: Designing multi-agent systems with zero supervision. *SEA@NeurIPS*, 2025b.

Ke, Z., Ming, Y., Xu, A., Chin, R., Nguyen, X.-P., Jwalapuram, P., Yavuz, S., Xiong, C., and Joty, S. Mas-orchestra: Understanding and improving multi-agent reasoning through holistic orchestration and controlled benchmarks, 2026. Preprint; Work in Progress.

Kim, Y., Gu, K., Park, C., Park, C., Schmidgall, S., Heydari, A. A., Yan, Y., Zhang, Z., Zhuang, Y., Malhotra, M., Liang, P. P., Park, H. W., Yang, Y., Xu, X., Du, Y., Patel, S., Althoff, T., McDuff, D., and Liu, X. Towards a science of scaling agent systems, 2025. URL https://arxiv.org/abs/2512.08296.

Krumdick, M., Lovering, C., Reddy, V., Ebner, S., and Tanner, C. No free labels: Limitations of llm-as-a-judge without human grounding, 2025. URL https://arxiv.org/abs/2503.05061.

LangChain. Open deep research github. 2025. URL https://github.com/langchain-ai/open_deep_research.

Liang, T., He, Z., Jiao, W., Wang, X., Wang, Y., Wang, R., Yang, Y., Shi, S., and Tu, Z. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*, 2023.

Liu, C. Y., Zeng, L., Xiao, Y., He, J., Liu, J., Wang, C., Yan, R., Shen, W., Zhang, F., Xu, J., et al. Skywork-reward-v2: Scaling preference data curation via human-ai synergy. *arXiv preprint arXiv:2507.01352*, 2025a.

Liu, R., Gao, J., Zhao, J., Zhang, K., Li, X., Qi, B., Ouyang, W., and Zhou, B. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling, 2025b. URL https://arxiv.org/abs/2502.06703.

Liu, Z., Zhang, Y., Li, P., Liu, Y., and Yang, D. A dynamic llm-powered agent network for task-oriented agent collaboration. *arXiv preprint arXiv:2310.02170*, 2024.

Mialon, G., Fourrier, C., Wolf, T., LeCun, Y., and Scialom, T. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

Panickssery, A., Bowman, S. R., and Feng, S. Llm evaluators recognize and favor their own generations, 2024. URL https://arxiv.org/abs/2404.13076.

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Shao, Z., Luo, Y., Lu, C., Ren, Z., Hu, J., Ye, T., Gou, Z., Ma, S., and Zhang, X. Deepseekmath-v2: Towards self-verifiable mathematical reasoning. *arXiv preprint arXiv:2511.22570*, 2025.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling test-time compute optimally can be more effective than scaling LLM parameters. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Wang, P.-Y., Liu, T.-S., Wang, C., Wang, Y.-D., Yan, S., Jia, C.-X., Liu, X.-H., Chen, X.-W., Xu, J.-C., Li, Z., and Yu, Y. A survey on large language models for mathematical reasoning, 2025. URL https://arxiv.org/abs/2506.08446.

Xu, A., Nguyen, X.-P., Zhou, Y., Wu, C.-S., Xiong, C., and Joty, S. Foundational automatic evaluators: Scaling multi-task generative evaluator training for reasoning-centric domains. *arXiv preprint arXiv:2510.17793*, 2025.

Zhang, G., Niu, L., Fang, J., Wang, K., BAI, L., and Wang, X. Multi-agent architecture search via agentic supernet. In *Forty-second International Conference on Machine Learning*, 2025a. URL https://openreview.net/forum?id=imcyVlzpXh.

Zhang, J., Xiang, J., Yu, Z., Teng, F., Chen, X.-H., Chen, J., Zhuge, M., Cheng, X., Hong, S., Wang, J., Zheng, B., Liu, B., Luo, Y., and Wu, C. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=z5uVAKwmjf.

Zhang, L., Hosseini, A., Bansal, H., Kazemi, M., Kumar, A., and Agarwal, R. Generative verifiers: Reward modeling as next-token prediction, 2025c. URL https://arxiv.org/abs/2408.15240.

Zhang, Z., Zheng, C., Wu, Y., Zhang, B., Lin, R., Yu, B., Liu, D., Zhou, J., and Lin, J. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*, 2025d.

Zhou, H., Wan, X., Sun, R., Palangi, H., Iqbal, S., Vuli, I., Korhonen, A., and Arık, S. Multi-agent design: Optimizing agents with better prompts and topologies, 2025a. URL https://arxiv.org/abs/2502.02533.

Zhou, Y., Xu, A., Wang, P., Xiong, C., and Joty, S. Evaluating judges as evaluators: The jetts benchmark of llm-as-judges as test-time scaling evaluators. *arXiv preprint arXiv:2504.15253*, 2025b.

Appendix

In Appendix A, we provide our implementation details of the experiments, including:

- **dataset details** (Appendix A.1),
- **baseline details** (Appendix A.2),
- **prompt templates** used in LLM reasoning (Appendix A.3),
- and **code implementation of MAS-ProVe** (Appendix A.4).

In Appendix B, we present additional empirical results, including:

- **extended experiments of MAS-Zero** (Appendix B.5),
- additional **benchmark analysis** (Appendix B.6)
- additional **analysis on context** (Appendix B.7), and
- **case study** (Appendix B.8).

A. Implementation Details

A.1. Datasets

We evaluate our framework on three distinct benchmarks, namely AIME24, AIME25 and GAIA. Both AIME24 and AIME25 comprise of 30 questions each with the information extraction subset of GAIA containing 103 questions.

AIME 24 & 25 (Mathematical Reasoning).
For the AIME datasets, we define difficulty empirically. We run a standard Chain-of-Thought (CoT) baseline 30 times per question. Questions are then stratified into *Easy*, *Medium*, and *Hard* strata based on the baseline's success rate, as detailed in Table 6. This ensures that our "Hard" category represents tasks that are stochastically unstable or consistently failing under standard inference.

Table 6. **Benchmark Difficulty Distribution.**

Dataset	Avg. Pass Rate (CoT Baseline)	Easy (>20/30)	Medium (11–20/30)	Hard (0–10/30)
AIME 24	58.3%	13 (43.3%)	9 (30.0%)	8 (26.7%)
AIME 25	42.0%	10 (33.3%)	4 (13.3%)	16 (53.3%)

GAIA (Information Extraction).
For the General AI Assistant benchmark (GAIA), we utilize the validation set of the *Information Extraction* subset ($N = 103$). Unlike AIME, GAIA difficulty is intrinsic, pre-annotated by the benchmark creators into Levels 1–3 based on the number of steps and tools required. As shown in Table 7, this dataset is heavily skewed towards multi-step tool use, with 90.3% of tasks requiring external tool calls (e.g., web search, file processing), making it an ideal testbed for evaluating process verification in tool-augmented settings.

Table 7. **GAIA Statistics.**

Metric	Level 1	Level 2	Level 3	Total
Count	39	52	12	103
Tool Requirements: • **Tool Required:** 93 tasks (90.3%) • **No Tools:** 10 tasks (9.7%)				

A.2. Baselines

Table 8 details the specific experimental protocols for each baseline.

- **Debate** (Du et al., 2024): We adopt the official LLM-Debate code by using the unified CoT prompt for initial thought generation, as in this paper, and also explicitly instructing each agent to refine its response based on the peer agent

responses. We run two agents for three iterations of debate, and for fair comparison, randomly select one of the final-round answers as the output.

- **DyLAN** (Liu et al., 2024): The Dynamic LLM-Powered Agent Network is re-implemented for Mathematical reasoning tasks by adopting from the MATH benchmark prompts. Tool calling is enabled with the usage of the DuckduckGo websearch call while dealing with GAIA benchmark with the roles of the agents adopted from MMLU implementation in the existing codebase.
- **AFlow** (Zhang et al., 2025b) and **MaAS** (Zhang et al., 2025a): For mathematical reasoning benchmarks, we retained the operator set and prompt libraries originally designed for the MATH benchmark, applying only minimal schema alignments to handle the distinct output formats of AIME. For the **GAIA** benchmark, we significantly re-engineered the agent space:
  - **Tool Integration:** We integrated a **DuckDuckGo** search tool directly into the execution environment, enabling agents to query external information required for open-ended assistant tasks.
  - **Prompt Adaptation:** We adopted the evaluation prompt templates from the official GAIA implementation to ensure consistent instruction adherence.
  - **Operator Pruning:** For the GAIA Information Extraction subset, we excluded the `Programmer` operator from the search space, focusing the architecture on retrieval and linguistic synthesis rather than code execution.
- **MAS-Zero** (Ke et al., 2025b): As the original implementation of MAS-Zero natively supports **AIME 24**, we extended this configuration to **AIME 25** without modification, utilizing the same prompt structure and atomic baselines. For **GAIA**, we expanded the architecture's fundamental building blocks. In addition to the standard four atomic baselines (e.g., CoT, Reflexion), we introduced a specialized **WebSearch Agent** primitive. This new block combines a DuckDuckGo retrieval interface with a Chain-of-Thought (CoT) wrapper, allowing the generated code to not only fetch results but also reason over the retrieved context before integrating it into the solution path.
- **ADAS** (Hu et al., 2025): Similar to MAS-Zero, the ADAS implementation required prompt refinement to handle AIME24 and AIME25. For **GAIA**, we applied the same architectural augmentation as described for MAS-Zero: the meta-agent was provided access to the **WebSearch Agent** primitive, effectively expanding the search space to include tool-augmented reasoning policies suitable for real-world information extraction.

For optimization-centric frameworks like AFlow, MaAS, and ADAS, we employ a strict validation/test split. Additionally, for generative architectures (MAS-Zero, ADAS), we restrict the primary evaluation to minimal iteration settings (1 and 5 iterations, respectively) and omit fixed baseline blocks (e.g., CoT, CoT-SC, Reflexion, etc.) during evaluation. This design choice isolates the performance of the generated agents and manages the exponential token growth inherent to these methods (see Table 5). Extended scalability results for MAS-Zero are provided in Appendix B.

Table 8. **Summary of MAS Baseline Configurations.** We distinguish between architectures evaluated on the full dataset versus those requiring optimization splits. Iteration limits for ADAS and MAS-Zero are imposed to balance computational feasibility with search depth.

Baseline	Protocol	Configuration Details
Debate	Full Dataset	2 Agents, 3 Rounds
DyLAN	Full Dataset	4 Agents, $T = 4$ Rounds
AFlow	Val/Test Split	20 Optimization Rounds, 2 Validation Rounds
MaAS	Train/Test Split	Sample Size $N = 4$
ADAS	Val/Test Split	5 Iterations. Additional WebSearch Baseline for GAIA
MAS-Zero	Full Dataset	1 Iteration. Additional WebSearch Baseline for GAIA

A.3. Prompt Templates

This section presents the prompt templates for the various judges used in our verification framework.

LLM-as-a-Judge

Please act as an impartial judge and evaluate the quality of multiple responses provided by AI assistants to the user prompt displayed below. You will be given several candidate responses.

Your task is to rank these responses from best to worst based on their correctness, reasoning quality, and completeness.

Consider: - Correctness of the final answer - Quality and clarity of reasoning - Completeness of the solution - Logical consistency

Be as objective as possible. Avoid any biases such as length or stylistic elements like formatting.

Before providing your ranking, think through the evaluation process and output your thoughts as an explanation.

After providing your explanation, you must output the ranking as a comma-separated list of candidate numbers (e.g., "2,1,3" means candidate 2 is best, candidate 1 is second, candidate 3 is worst). Please enclose your ranking in <ranking> and </ranking> tags.

FARE-20B

PROMPT_RANKING_SYSTEM = """ Please act as an impartial judge and evaluate the quality of the responses provided by three AI assistants to the user prompt displayed below. You will be given assistant A's answer, assistant B's answer, and assistant C's answer. Your job is to determine which assistant's answer is better. If assistant A is better, output [A]. If assistant B is better, output [B]. If assistant C is better, output [C].

Here are some rules for evaluation (1) When evaluating the assistants' answers, identify any mistakes or inaccurate information. Focus on the content each response and select the response that is logically sound and error free. (2) If both responses contain inaccurate information, select the response that arrives at the correct response (3) Avoid any biases, such as order of responses, length, or stylistic elements like formatting

Before outputting your final judgment, provide an explanation of your judgment. Your explanation should discuss why your chosen response is better based on the evaluation criteria. The explanation should concretely discuss strengths and weaknesses of both answers. After outputting your explanation, provide your final judgment. Use the following format: Explanation: Your explanation here Verdict: Your final verdict """.strip()

PROMPT_PAIRWISE = """ [User Question] instruction

[The Start of Assistant A's Answer] {response_a} [The End of Assistant A's Answer]

[The Start of Assistant B's Answer] {response_b} [The End of Assistant B's Answer]

[The Start of Assistant C's Answer] {response_c} [The End of Assistant C's Answer] """.strip()

Summarizer Prompt for PRM/ RM/ FARE-20B/ GPT-5-mini

You are a summarization assistant for multi-agent system processes.

Your task is to summarize the given context into a concise and informative summary. The total length of the summary should be fewer than {max_context_length} tokens.

IMPORTANT PRINCIPLES: - This is NOT style normalization. - This is NOT reformatting to a fixed template. - The cleaned solution should look like the SAME author wrote it, just more concisely.

STRICT REQUIREMENTS: 1) Faithfulness: - Do NOT change the meaning. - Do NOT change the final answer(s). 2) Reasoning preservation: - Keep ALL logical reasoning steps and ALL necessary mathematical derivations. - You may remove repeated steps, detours, or restatements ONLY if the reasoning remains intact. 3) Language & style preservation: - Preserve the original narrative tone (e.g. exploratory, explanatory, corrective). - Preserve structural choices used in the original solution (such as step labels, section headers, or paragraph structure), unless they are clearly redundant. 4) Formatting: - Do NOT convert narrative explanations into bullet fragments or terse notes. - Do NOT collapse structured explanations into formula-only derivations. - Keep LaTeX for mathematics.

Output ONLY the cleaned solution text. No extra commentary.

General prompt for GAIA

You are a general AI assistant with access to web search capabilities. If you are asked for a number, don't use comma to write your number neither use units such as $ or percent sign unless specified otherwise. If you are asked for a string, don't use articles, neither abbreviations (e.g. for cities), and write the digits in plain text unless specified otherwise. If

you are asked for a comma separated list, apply the above rules depending of whether the element to be put in the list is a number or a string.

QUESTION: {question}

If you need to look up information to answer this question, you can use the web_search function available to you. Call it with specific search queries when you need factual information, current data, or verification.

You can freely reason in your response, please: 1. include the thinking process in your response, in a detailed and extended manner, and after that, 2. enclose the final answer within <answer></answer> tags

A.4. Code Templates

Here we will define the code templates that serve the primary focus of our process verification framework. We will go over the parallel search decorator that acts as the middleware between the client(MAS) and the Server(Judge). We also show the MASBase main class which helps in managing the context states and also serve as an independent handler of the flow of process verification.

Parallel Search Decorator

```
def llm_parallel_search_decorator(llm_func):
    """
    Decorator to run the LLM call function MAX_PARALLEL_SEARCH_CALLS times in parallel
    ,
    send the results to the server via client, and pick the best response.
    """
    async def gather_calls(*args, **kwargs):
        # Call the original LLM func in parallel
        tasks = [llm_func(*args, **kwargs) for _ in range(MAX_PARALLEL_SEARCH_CALLS)]
        responses = await asyncio.gather(*tasks)
        return responses

    def send_to_server(responses, client: BaseClient, task_type=None):
        # Sends all candidate responses to server (expects 'server_client' conforming
    to your client API)
        return client.send_request(task_type, "judge", responses)

    def decorator_wrapper(*args, **kwargs):
        """
        kwargs must include 'client', and optionally 'task_type'
        """
        task_type = kwargs.pop('task_type', None)
        client = kwargs.pop('client', None)
        if client is None:
            raise ValueError("A 'client' must be provided as a kwarg to the decorated
    function.")

        # Run the LLM calls in parallel & collect results
        responses = asyncio.run(gather_calls(*args, **kwargs))

        # Send all results to the server judge and get rankings
        server_result = send_to_server(responses, client, task_type)
        rankings = server_result.get("rankings", list(range(len(responses))))
        best_idx = rankings[0] if rankings else 0

        return responses[best_idx]

    return decorator_wrapper
```

MASBase Class

```
class MASBase:
    def __init__(self):
        self.trajectory = []
        self.trajectory_lock = asyncio.Lock()

    @property
    def progress(self):
        return json.dumps({f"step_{i+1}": step for i, step in enumerate(self.
    trajectory)}, indent=4)

    @staticmethod
    def update_trajectory(func):
        """
        Appends the latest step/result to the trajectory and updates the progress.
        Should be called automatically by wrappers of generation calls.
        """
        async def wrapper(self, *args, **kwargs):
            # Get the original result from the decorated function
            result = await func(self, *args, **kwargs)

            # Extract content for trajectory while preserving original result
            trajectory_content = result
            if isinstance(result, tuple) or isinstance(result, list):
                trajectory_content = result[0]
            # Convert dict to string for trajectory storage
            if isinstance(trajectory_content, dict):
                trajectory_content = json.dumps(trajectory_content)

            async with self.trajectory_lock:
                self.trajectory.append(trajectory_content)

            return result
        return wrapper

    def run(self):
        raise NotImplementedError("Subclasses must implement this method")
```

MASBase Adaptation on AFlow for Agent Call Process Verification

```
class MASAFlow(MASBase):
    """
    MAS wrapper for AFlow workflows - analogous to MASDebate for debate.
    Wraps workflow execution with trajectory tracking and process verification.

    Usage (parallel to MASDebate):
        mas_aflow = MASAFlow(workflow, problem, expected_answer, task_type)
        result = await mas_aflow.run()
    """

    def __init__(self, workflow, task_type: str = "math"):
        super().__init__()
        self.workflow = workflow
        self.task_type = task_type
        self.problem = ""  # Will be set per call

        # Inject decorators into workflow operators (only once)
        self._inject_decorators()
```

```
    @MASBase.update_trajectory
    @llm_parallel_search_decorator
    async def _call_custom(self, *args, **kwargs):
        """Decorated custom operator call"""
        result = await self._original_custom(*args, **kwargs)
        return result

    @MASBase.update_trajectory
    @llm_parallel_search_decorator
    async def _call_programmer(self, *args, **kwargs):
        """Decorated programmer operator call"""
        return await self._original_programmer(*args, **kwargs)

    @MASBase.update_trajectory
    @llm_parallel_search_decorator
    async def _call_sc_ensemble(self, *args, **kwargs):
        """Decorated sc_ensemble operator call"""
        return await self._original_sc_ensemble(*args, **kwargs)

    @MASBase.update_trajectory
    @llm_parallel_search_decorator
    async def _call_answer_generate(self, *args, **kwargs):
        """Decorated answer_generate operator call"""
        return await self._original_answer_generate(*args, **kwargs)

    async def __call__(self, problem: str):
        """
        Execute the workflow with trajectory tracking for a specific problem.
        Returns same format as normal workflow: (final_answer, cost)
        """
        # Update problem for this execution
        self.problem = problem
        # Reset trajectory for new problem
        self.trajectory = []
        # Execute workflow
        result = await self.workflow(problem)
        return result

    async def run(self, problem: str):
        """Alias for __call__ for backwards compatibility"""
        return await self(problem)
```

Iteration Level Process Verification on AFlow Workflow

```
async def __call__(self, problem: str):
        """
        Wrapper that triggers the parallel search.
        The decorator runs the logic 3 times in parallel and the Judge picks the best.
        """
        # Call the decorated function with required Judge parameters
        best_result = await self._run_logic_async(
            problem=problem,
            question=problem,       # REQUIRED: User prompt for the Judge
            task_type="math"    # REQUIRED: Triggers the Math/QA prompt template
        )

        # Unpack the best result
        if isinstance(best_result, dict):
            return best_result["output"], best_result["cost"]
        else:
            return str(best_result), 0
```

```
# ------------------------------------------------------------------------
# 2. The Decorated Logic (Runs N times in Parallel)
# ------------------------------------------------------------------------
@llm_parallel_search_decorator
async def _run_logic_async(self, problem: str, **kwargs):
    """
    The core workflow logic.
    Runs N times in parallel. Returns a full trace for the Judge to evaluate.
    """
    import re

    # --- Step 1: Generate Candidates ---
    # First candidate: concise final numeric answer
    instr1 = prompt_custom.CANDIDATE_PROMPT_1
    resp1 = await self.custom(input=problem, instruction=instr1)

    # Second candidate: step-by-step
    instr2 = prompt_custom.CANDIDATE_PROMPT_2
    resp2 = await self.custom(input=problem, instruction=instr2)

    # --- Step 2: Ensemble ---
    # Handle dynamic method name check from your snippet
    solutions = [resp1['response'], resp2['response']]
    if hasattr(self, 'sc_ensemble'):
        ensemble = await self.sc_ensemble(solutions=solutions, problem=problem)
    else:
        ensemble = await self.sc(solutions=solutions, problem=problem)

    chosen = ensemble['response']

    # --- Step 3: Extraction ---
    nums = re.findall(r'\b-?\d+\b', chosen)
    candidate_num = nums[-1] if nums else ''

    # --- Step 4: Programmer Verification ---
    verification_log = ""
    prog_verify = await self.programmer(problem=problem, analysis=f"Verify whether
 the final numeric answer is {candidate_num}. Return only the final integer if
verified, otherwise return a different integer computed.")
    verified = prog_verify.get('output', '').strip()
    verification_log = f"Verification Output: {verified}"

    if verified and re.search(r'\b-?\d+\b', verified):
        final_answer = verified
    else:
        # Fallback: compute from scratch
        prog_full = await self.programmer(problem=problem, analysis="Compute and
return only the final numeric answer (integer).")
        final_answer = prog_full.get('output', '').strip() or candidate_num
        verification_log += f"\nFallback Calculation: {final_answer}"

    # Get total cost
    total_cost = self.llm.get_usage_summary().get("total_cost", 0)

    # --- Construct Trace for the Judge ---
    # This is what allows the Judge to rank this attempt against the parallel
others
    trace_str = (
        f"--- Candidate 1 (Concise) ---\n{resp1.get('response', '')}\n\n"
        f"--- Candidate 2 (Step-by-Step) ---\n{resp2.get('response', '')}\n\n"
        f"--- Ensemble Decision ---\n{chosen}\n\n"
        f"--- Programmer Verification ---\n{verification_log}\n\n"
```

```
            f"--- Final Answer ---\n{final_answer}"
        )

        # --- Return Dictionary for Decorator ---
        return {
            "response": trace_str,      # The reasoning trace (Primary for Judge)
            "output": final_answer,
            "cost": total_cost
        }
```

Since all the Baselines used have their own prompt templates presented in their own paper, we will look to address the templates that included drastic changes from the original which we had to implement such as the WebSearch agent for ADAS and MAS-Zero.

WebSearch Agent for ADAS and MAS-Zer0

```
async def forward(self, taskInfo, extra_info):
    # Instruction for the WebSearch with Chain-of-Thought approach
    # The agent will execute web search and use results for reasoning
    cot_instruction = self.cot_instruction

    # Use the original question/task as the search query
    # Note: The meta-agent can modify this search_query in generated code
    # for further iterations
    # or customize it during code generation if needed
    search_query = taskInfo.content.strip()

    print(f"WebSearch agent using query: {search_query}")

    # Execute web search with the generated query
    web_search_results = "Web search unavailable."
    try:
        from ddgs import DDGS

        with DDGS() as ddgs:
            results = list(ddgs.text(search_query, max_results=5))

        formatted_output = "Search results:\n"
        for i, result in enumerate(results, 1):
            title = result.get('title', 'Untitled')
            link = result.get('href', '')
            snippet = result.get('body', 'No content available.')
            formatted_output += f"--- SOURCE {i}: {title} ---
            \nURL: {link}\n\nCONTENT:\n{snippet}\n\n"

        web_search_results = formatted_output
        print(f"Web search completed - found {len(results)} results")
    except Exception as e:
        print(f"Web search failed: {e}")

    # Add web search results to the instruction instead of extra_info
    enhanced_instruction = f"{cot_instruction}\n\n### Web Search Results
    ###\n{web_search_results}\n\nUse the above search results to help answer
    the question."

    # Instantiate a new LLM agent specifically for CoT with web search
    websearch_agent = LLMAgentBase(['thinking', 'answer'], 'WebSearch Agent'
    , model=self.node_model, temperature=0.0)

    # Prepare the inputs for the CoT agent
    cot_agent_inputs = [taskInfo]
```

```
    # Get the response from the CoT agent with enhanced instruction
    thinking, answer = await websearch_agent(cot_agent_inputs
    , extra_info, enhanced_instruction)
    final_answer = self.make_final_answer(thinking, answer)

    # Return only the final answer
    return final_answer
```

Generated Code for MAS-Zero with WebSearch Agent

```
 Problem: "What integer-rounded percentage of the total length of the harlequin shrimp
     recorded in Omar Valencfia-Mendez 2017 paper was the sea star fed to the same
    type of shrimp in G. Curt Fiedler's 2002 paper?"

async def forward(self, taskInfo, extra_info):
    from collections import Counter
    sub_tasks = []
    agents = []

    # --- Sub-task 1: WebSearch for Omar Valencia-Mendez 2017 harlequin shrimp total
    length ---
    # Implement WebSearch block (as in archive)
    search_query_1 = f"Omar Valencia-Mendez 2017 harlequin shrimp total length"
    web_search_results_1 = "Web search unavailable."
    try:
        from ddgs import DDGS
        with DDGS() as ddgs:
            results1 = list(ddgs.text(search_query_1, max_results=5))
        formatted_output_1 = "Search results:\n"
        for i, result in enumerate(results1, 1):
            title = result.get('title', 'Untitled')
            link = result.get('href', '')
            snippet = result.get('body', 'No content available.')
            formatted_output_1 += f"--- SOURCE {i}: {title} ---\nURL: {link}\n\
    nCONTENT:\n{snippet}\n\n"
        web_search_results_1 = formatted_output_1
    except Exception:
        web_search_results_1 = "Web search failed or ddgs unavailable."

    # Self-Consistency: run N CoT agents to extract numeric shrimp total length
    N1 = getattr(self, 'max_sc', 5) if hasattr(self, 'max_sc') else 5
    cot_instruction_1 = (
        "Sub-task 1: Based on the provided web search results and the original
    question, extract the reported total length of harlequin shrimp from the Omar
    Valencia-Mendez 2017 paper. "
        "Steps: 1) Use the search results to find the numeric total length and its
    unit (e.g., mm or cm). 2) If multiple values appear, choose the value explicitly
    described as 'total length' or 'TL' for harlequin shrimp. 3) Output the numeric
    value and unit clearly."
        f"\n\n### Web Search Results ###\n{web_search_results_1}\n\n"
    )
    cot_agents_1 = [LLMAgentBase(['thinking', 'answer'], 'Chain-of-Thought Agent',
    model=self.node_model, temperature=0.5) for _ in range(N1)]

    thinking_mapping_1 = {}
    answer_mapping_1 = {}
    possible_answers_1 = []
    for i in range(N1):
        thinking_i, answer_i = await cot_agents_1[i]([taskInfo], extra_info,
    cot_instruction_1, is_sub_task=True)
```

```
    agents.append(f'CoT_sc_paper1 agent {cot_agents_1[i].id}, round {i}, thinking:
 {thinking_i.content}; answer: {answer_i.content}')
    possible_answers_1.append(answer_i.content)
    thinking_mapping_1[answer_i.content] = thinking_i
    answer_mapping_1[answer_i.content] = answer_i

def majority_voting(answers):
    return Counter(answers).most_common(1)[0][0]

chosen_answer_1_text = majority_voting(possible_answers_1)
thinking_1 = thinking_mapping_1[chosen_answer_1_text]
answer_1 = answer_mapping_1[chosen_answer_1_text]
sub_tasks.append(f"Sub-task 1 output: thinking - {thinking_1.content}; answer - {
answer_1.content}")

# --- Sub-task 2: WebSearch for G. Curt Fiedler 2002 sea star fed length ---
search_query_2 = f"G. Curt Fiedler 2002 harlequin shrimp sea star fed length"
web_search_results_2 = "Web search unavailable."
try:
    from ddgs import DDGS
    with DDGS() as ddgs:
        results2 = list(ddgs.text(search_query_2, max_results=5))
    formatted_output_2 = "Search results:\n"
    for i, result in enumerate(results2, 1):
        title = result.get('title', 'Untitled')
        link = result.get('href', '')
        snippet = result.get('body', 'No content available.')
        formatted_output_2 += f"--- SOURCE {i}: {title} ---\nURL: {link}\n\
nCONTENT:\n{snippet}\n\n"
    web_search_results_2 = formatted_output_2
except Exception:
    web_search_results_2 = "Web search failed or ddgs unavailable."

# Self-Consistency ensemble to extract sea star length
N2 = getattr(self, 'max_sc', 5) if hasattr(self, 'max_sc') else 5
cot_instruction_2 = (
    "Sub-task 2: Based on the provided web search results and the original
question, extract the reported length of the sea star fed to the harlequin shrimp
in G. Curt Fiedler's 2002 paper. "
    "Steps: 1) Use the search results to find the numeric length and its unit. 2)
Prefer explicit numeric measurements tied to the fed sea star specimen. 3) Output
the numeric value and unit clearly."
    f"\n\n### Web Search Results ###\n{web_search_results_2}\n\n"
)
cot_agents_2 = [LLMAgentBase(['thinking', 'answer'], 'Chain-of-Thought Agent',
model=self.node_model, temperature=0.5) for _ in range(N2)]

thinking_mapping_2 = {}
answer_mapping_2 = {}
possible_answers_2 = []
for i in range(N2):
    thinking_i, answer_i = await cot_agents_2[i]([taskInfo], extra_info,
cot_instruction_2, is_sub_task=True)
    agents.append(f'CoT_sc_paper2 agent {cot_agents_2[i].id}, round {i}, thinking:
 {thinking_i.content}; answer: {answer_i.content}')
    possible_answers_2.append(answer_i.content)
    thinking_mapping_2[answer_i.content] = thinking_i
    answer_mapping_2[answer_i.content] = answer_i

chosen_answer_2_text = majority_voting(possible_answers_2)
thinking_2 = thinking_mapping_2[chosen_answer_2_text]
answer_2 = answer_mapping_2[chosen_answer_2_text]
sub_tasks.append(f"Sub-task 2 output: thinking - {thinking_2.content}; answer - {
```

```
answer_2.content}")

# --- Sub-task 3: Compute percentage and round ---
cot_final_instruction = (
    "Sub-task 3: Based on the outputs from sub-task 1 and sub-task 2, compute the
percentage that the sea star length is of the total harlequin shrimp length. "
    "Steps: 1) Extract numeric values and units from the two answers provided. 2)
If units differ, convert so both are same (e.g., mm). 3) Compute percentage = (sea
 star length / shrimp total length) * 100. 4) Round to nearest integer and output
only the integer percentage along with a one-line justification of unit conversion
 and arithmetic."
)
final_cot_agent = LLMAgentBase(['thinking', 'answer'], 'Chain-of-Thought Agent',
model=self.node_model, temperature=0.0)

thinking_final, answer_final = await final_cot_agent(
    [taskInfo, thinking_1, answer_1, thinking_2, answer_2],
    extra_info,
    cot_final_instruction,
    is_sub_task=True
)
agents.append(f'Final CoT agent {final_cot_agent.id}, thinking: {thinking_final.
content}; answer: {answer_final.content}')
sub_tasks.append(f"Sub-task 3 output: thinking - {thinking_final.content}; answer
- {answer_final.content}")

# Make final answer with tracked sub_tasks and agents
final_answer = self.make_final_answer(thinking_final, answer_final, sub_tasks,
agents)
return final_answer
```

B. Additional Experimental Results

B.1. Computational Overhead of Process Verification

To quantify the computational overhead introduced by the MAS-ProVe framework, we aggregated the average wall-clock execution times on the AIME benchmark to provide a real-world estimate of the verification cost. Table 9 presents a comparison of the baseline execution (without process verification) against configurations utilizing our Generative Judge and Reward Model (RM) across six different MAS frameworks.

Table 9. Average wall-clock execution time for baseline and verified configurations across different MAS frameworks on the AIME benchmark.

MAS Framework	Baseline (w/o Judge)	w/ Generative Judge	w/ Reward Model (RM)
Debate	$\sim$ 2m 39s	$\sim$ 8m 52s	$\sim$ 6m 35s
AFlow	$\sim$ 6m 55s	$\sim$ 9m 24s	$\sim$ 7m 12s
MaAS	$\sim$ 4m 32s	$\sim$ 7m 10s	$\sim$ 5m 47s
DyLAN	$\sim$ 7m 00s	$\sim$ 11m 50s	$\sim$ 9m 33s
ADAS (1 Iteration)	$\sim$ 9m 14s	$\sim$ 29m 17s	$\sim$ 21m 32s
MAS-Zero (1 Iteration)	$\sim$ 17m 21s	$\sim$ 37m 31s	$\sim$ 26m 51s

While these wall-clock measurements offer a practical perspective on system latency, they may not perfectly reflect the true computational overhead. Our evaluation relies on asynchronous calls to commercial LLM APIs, where response times can vary substantially due to external server load and organizational rate limits.

To provide a rigorous, hardware-agnostic measure of this computational cost, we formalize the verification overhead directly through token consumption and API call volume. Building upon the methodology established in Table 4 of the main text – which tracks baseline reasoning steps and token consumption per MAS – the integration of process verification explicitly

scales the base cost. The additional overhead is mathematically bounded by the search width N (generating N parallel trajectories rather than 1) plus the total input and output tokens consumed by the verifier module when scoring the N partial trajectories. This token-based accounting isolates the intrinsic algorithmic cost of the MAS-ProVe framework from transient network fluctuations.

B.2. Analysis of Failure Modes: Verifier Misranking and Context Sensitivity

The performance degradations occasionally observed in our main results stem from two distinct forms of verifier misranking. While the failures of scoring-based Reward Models (RMs) or PRMs are primarily driven by out-of-domain distribution shifts, the Generative Judge's occasional degradations are heavily driven by context noise.

To empirically isolate and prove this context sensitivity, we conducted an ablation study on the AFlow framework using the AIME 2024 benchmark. We evaluated 153 branch comparisons, prompting the Generative Judge to rank the exact same candidate branches under three different context management formats: *Raw*, *Summarized*, and *Step-Only*.

Evaluating the pairwise consistency across these 153 trials yielded the following distribution:

- **All 3 formats agreed:** 56.2% (86/153)
- **Exactly 2 formats agreed:** 41.8% (Raw=Summ: 15.7%; Raw=Step: 13.7%; Summ=Step: 12.4%)
- **All 3 differed:** 2.0% (3/153)

While these results do not rule out search myopia entirely, they conclusively demonstrate the verifier's *brittleness* to context. In over 43% of the evaluated cases (67 out of 153), the Generative Judge altered its top-ranked selection based *purely* on how the history was formatted. Thus, verifier misranking driven by context sensitivity serves as a primary catalyst for performance degradation within verified multi-agent workflows.

B.3. Exhaustive Search Analysis: Generation Limits vs. Verifier Capacity

To support the findings discussed in the main text regarding the fundamental solvability ceiling, we designed an experiment to rigorously isolate the MAS's intrinsic generation capability from the Generative Judge's ranking capacity. We conducted an exhaustive search (Pass@128) on the Debate framework across four AIME questions of varying baseline difficulty.

By generating 128 trajectories per question, we established the MAS's intrinsic generation capability (Pass Rate). We then tested our Generative Judge on a static subset comprising one correct branch and two incorrect branches to evaluate its ranking success over 20 trials. The results are summarized in Table 10.

Table 10. MAS intrinsic pass rate vs. Generative Judge ranking success across 128 sampled trajectories on AIME.

Question	Correct / 128	Pass Rate	Judge Success
Q21 (AIME24)	0	0.0%	0.0% (No correct path)
Q12 (AIME25)	2	1.5%	45.0%
Q25 (AIME24)	13	10.1%	80.0%
Q27 (AIME24)	51	39.8%	100.0%

These results clearly delineate the boundary between generation failure and verification failure. For Q21, the MAS fails to generate a single correct solution across 128 attempts. Because the correct trajectory is never produced, the Judge inherently has a 0% success rate – illustrating a fundamental generation limit rather than an inadequate verifier. Conversely, for Q12, the MAS rarely finds the solution (1.5% pass rate), yet the Judge identifies it 45.0% of the time when present. This demonstrates that provided the trajectory exists within the search space, the verifier possesses the capacity to successfully guide the search process toward the correct reasoning path.

B.4. Impact of Search Heuristics: Beam Search Ablation

To evaluate the impact of utilizing more sophisticated search heuristics during process verification, we implemented a Beam Search (width = 3) on the Debate framework utilizing our optimal Generative Judge. Table 11 compares the performance of this expanded search strategy against our standard Greedy Search and the unverified baseline across the AIME 2024, AIME 2025, and GAIA benchmarks.

Table 11. Comparison of search strategies on the Debate framework across multiple benchmarks.

Benchmark	Baseline (No Verifier)	Greedy Search (Judge)	Beam Search (Judge, $w = 3$)
AIME 2024	74.13 ± 3.40	81.11 ± 1.92	82.72 ± 0.87
AIME 2025	61.87 ± 3.21	62.97 ± 3.38	66.67 ± 4.72
GAIA	31.07 ± 2.07	30.28 ± 1.51	30.07

While Beam Search provides a slight performance improvement on the mathematical reasoning tasks, the gains over our standard Greedy Search baseline are marginal. Although this ablation is limited to the Debate framework, it supports our broader hypothesis that more sophisticated search heuristics ultimately remain bounded by the MAS's intrinsic generation limits (as discussed in Appendix B.3). Consequently, these findings illustrate that our greedy heuristic offers a highly effective, practical trade-off between optimal verification performance and framework-agnostic integration.

B.5. MAS-Zero Extended Results

Here, we provide an extented tabulation from our Table 1 in providing the results of Process verification across different judge types on MAS-Zero with $N = 5$ iterations in Table 12. The results show that there is a more consistent improvement in performance across benchmarks, suggesting that scaling of this verification framework could possibly serve as something that could provide further surprising insights.

Table 12. Extended results of Process verification on MAS-Zero with 5-iterations.

Method	MAS-Arch	Proc-Eval Level	AIME24			AIME25			GAIA		
			Judge	RM	PRM	Judge	RM	PRM	Judge	RM	PRM
MAS-Zero	Adaptive	No ProVe	44.45$\pm$6.94			40.86$\pm$3.55			11.93$\pm$0.58		
		Iteration	58.34$\pm$2.35	51.67$\pm$7.07	46.69$\pm$5.75	44.42$\pm$8.37	50.00$\pm$4.71	48.34$\pm$2.35	23.20$\pm$2.89	29.50$\pm$3.11	11.59$\pm$3.43

B.6. Additional Benchmark Analysis

We also present additional results with respect to the question-wise analysis across Baseline, Judge and Best versions of Process Verification.

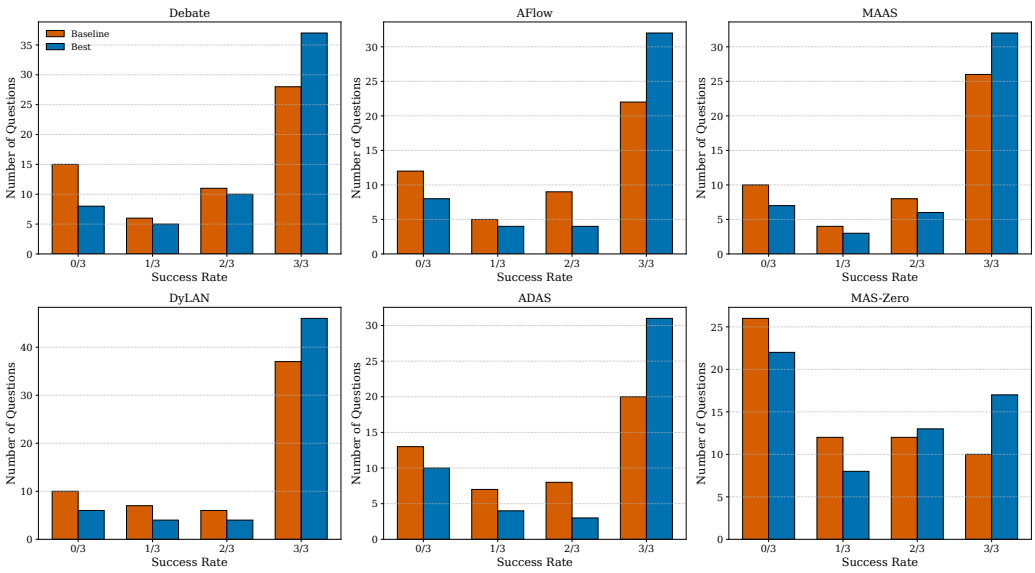

Figure 7. Shows the full distribution of outcomes (0/3, 1/3, 2/3, 3/3 successes) for each MAS on Mathemaical Reasoning Benchmarks

In contrast to mathematical reasoning, where summarization serves as an effective mechanism, results on the GAIA

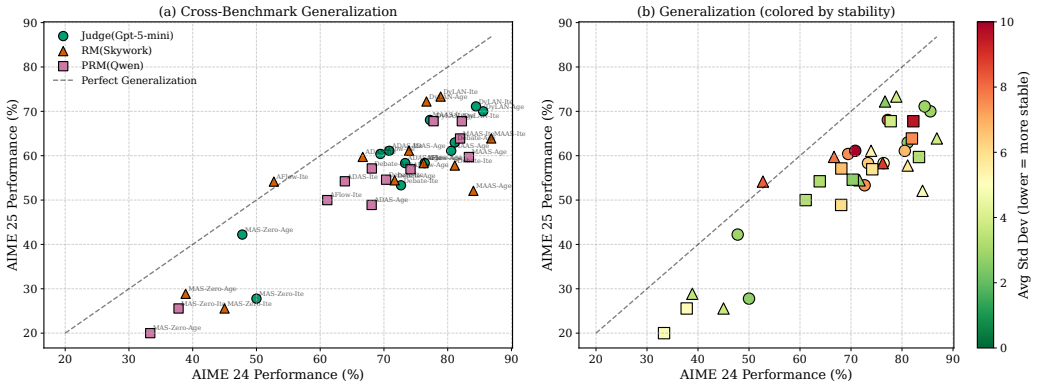

Figure 8. Cross-benchmark generalization comparing AIME 2024 and AIME 2025 accuracy. Each point represents a MAS-evaluator configuration, with the diagonal line indicating perfect generalization (equal performance on both benchmarks). Panel (a) shows configurations grouped by evaluation approach, while panel (b) encodes stability (standard deviation across trials) through color intensity. Points' positions relative to the diagonal reflect the relative performance difference between the two benchmark years.

benchmark reveal a different dynamic. As illustrated in Fig. 9, the *Summarized* context strategy (Red bars) underperforms compared to *Raw History* or *Step-Only* baselines across most architectures.

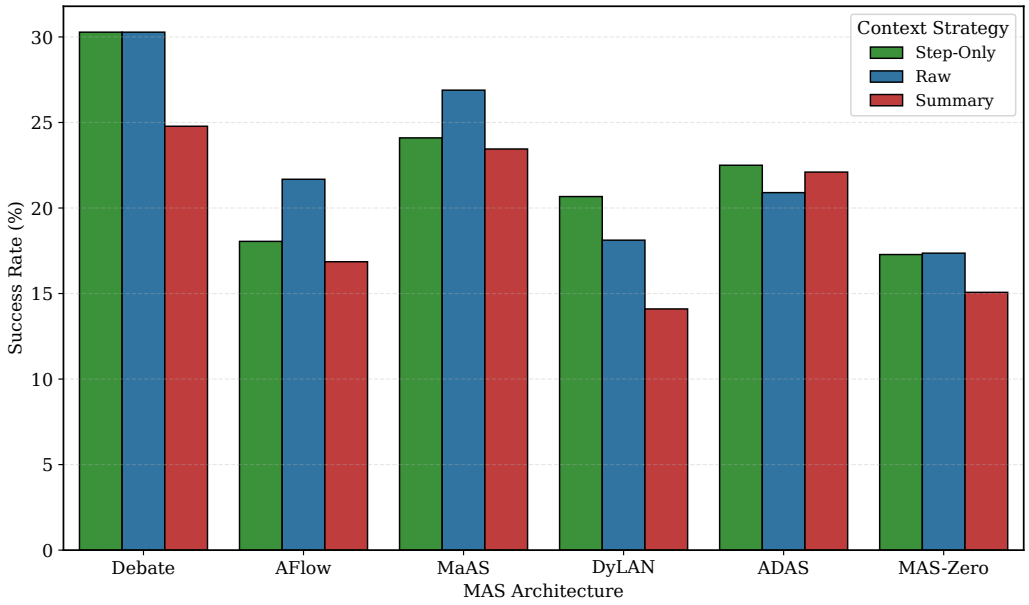

Figure 9. Unlike reasoning tasks, Information Extraction tasks suffer from summarization. The *Summarized* strategy (Red) frequently yields the lowest accuracy, suggesting that abstraction causes the loss of critical, granular details required for verifying tool-use and retrieval fidelity.

B.7. Additional Context Analysis

We define Growth Rate calculated as $\frac{\text{Tokens}_{\text{strategy}} - \text{Tokens}_{\text{Step-Only}}}{\text{Tokens}_{\text{Step-Only}}} \times 100$. Summary strategies show consistent 25% overhead across architectures, Summarized Context adds ∼35%, while Raw context imposes a substantial 270% average overhead with high variability across architectures.

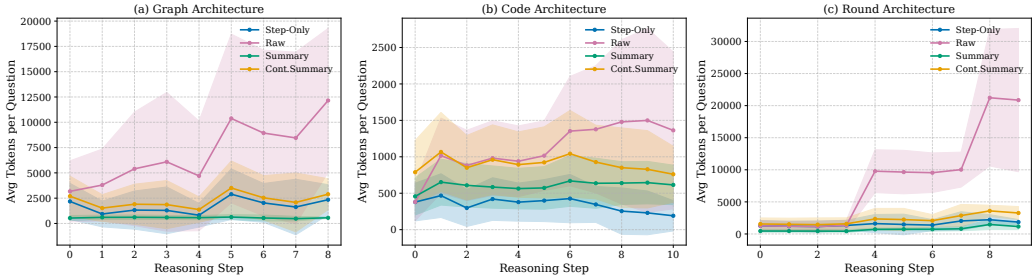

Figure 10. Token Progression Vs Average Reasoning Steps with respect to architecture. Graph Architecture represents *AFlow, MaAS*. Code architectures include *ADAS and MAS-Zero*. Round based architectures include *Debate and DyLAN*. The token overhead of *Raw History* scales linearly or quadratically with trajectory depth, the *Summarized* context maintains a consistent, low-variance cost profile across all architectures.

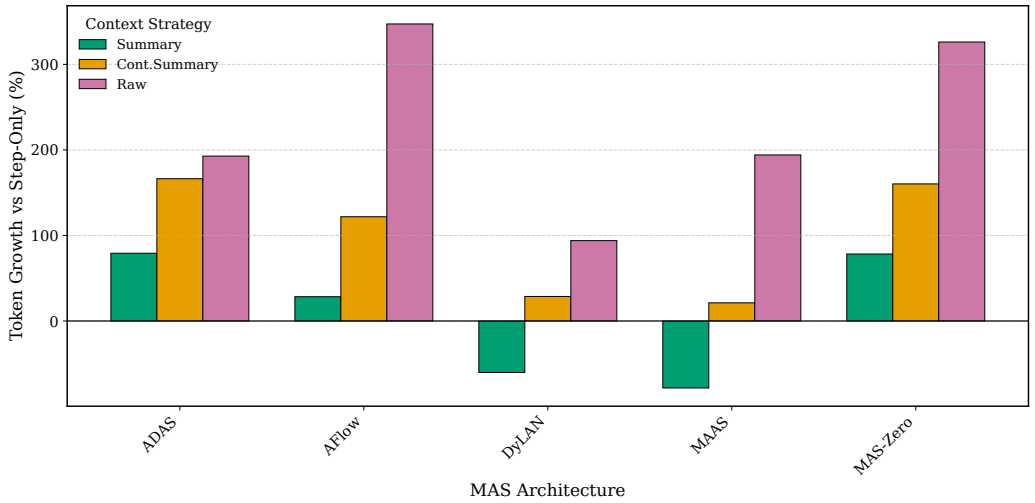

Figure 11. Token overhead by context strategy. Bars represent percentage increase in token consumption relative to Step-Only baseline for each MAS architecture. Summary adds ∼25% overhead, Continuous Summary ∼35%, and Raw context ∼270%. The zero line indicates Step-Only baseline performance. Colors: Summary (green), Continuous Summary (orange), Raw (purple).

Table 13. Average token growth rate by context strategy and architecture, calculated relative to Step-Only baseline. All values represent percentage increase in token consumption.

MAS Architecture	Summary (%)	Cont.Summary (%)	Raw (%)	Max Growth (%)
DyLAN	+23.5	+33.3	+261.1	+261.1
MAAS	+23.7	+34.8	+265.2	+265.2
AFlow	+26.4	+36.3	+272.2	+272.2
ADAS	+21.9	+30.5	+256.3	+256.3
MAS-Zero	+30.3	+38.6	+291.0	+291.0
Mean	**+25.2**	**+34.7**	**+269.2**	**+269.2**
Std Dev	**3.1**	**3.0**	**12.8**	**12.8**

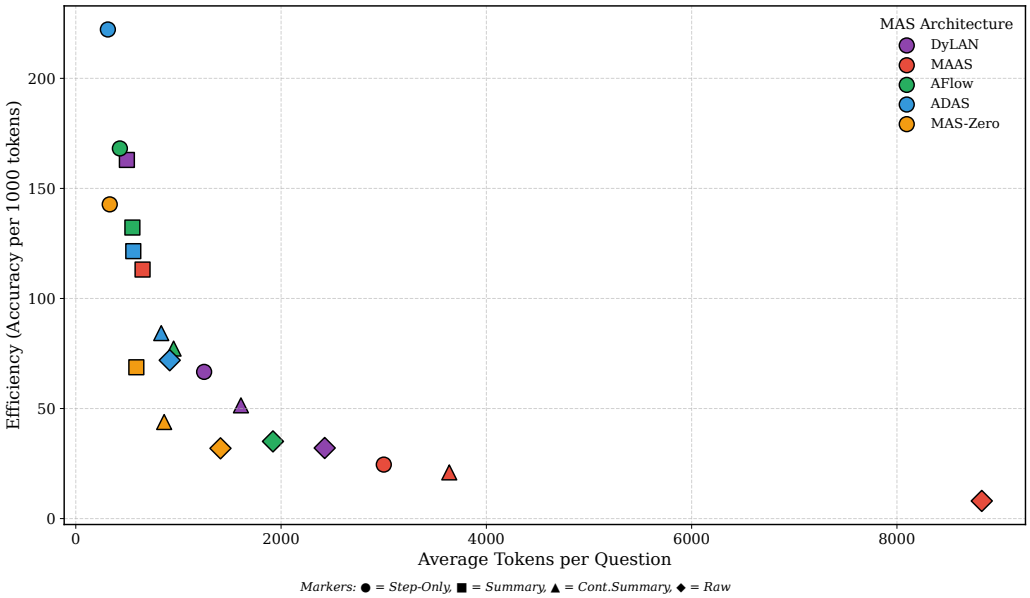

Figure 12. Token efficiency comparison across MAS architectures and context strategies. Efficiency is defined as $\eta = \frac{\text{AIME Accuracy (\%)}}{\text{Tokens}/1000}$, measuring accuracy points achieved per 1000 tokens. Colors represent MAS architectures (DyLAN, MAAS, AFlow, ADAS, MAS-Zero), while marker shapes indicate context strategies: (Circle)Step-Only, (Square)Summary, (Triangle) Continuous Summary, (Diamond)Raw. Step-Only consistently achieves the highest efficiency (33-39), while Raw context shows the lowest (7-10) despite consuming 3-4× more tokens. The scatter pattern reveals a clear inverse relationship between token consumption and efficiency, highlighting the diminishing returns of more comprehensive context strategies.

B.8. Case Study

Example 1: GPT-5-mini Judge Rationale

Question: Let A be the set of positive integer divisors of 2025. Let B be a randomly selected subset of A. The probability that B is a nonempty set with the property that the least common multiple of its elements is 2025 is $\frac{m}{n}$, where m and n are relatively prime positive integers. Find $m+n$.

Context for all 3 candidates:

- **Step 1:** Prime factorization: $2025 = 45^2 = (9 \cdot 5)^2 = 3^4 \cdot 5^2$. Primes 3 and 5 with exponents 4 and 2. Each positive divisor has form $3^a \cdot 5^b$ with $0 \le a \le 4$ and $0 \le b \le 2$. Total divisors = $(4+1)(2+1) = 15$.
- **Step 2:** Every positive divisor d of 2025 can be written as $d = 3^a \cdot 5^b$. Thus there are 15 positive divisors.

Candidate 1 Response:
"We must count for each prime p (3 with $e_{max} = 4$, 5 with $e_{max} = 2$) how many divisors have exponent exactly k. [...] For prime 3, number of divisors with exponent exactly k equals the number of choices for the other prime's exponent. [...] Thus per-prime counts are: for $p = 3$ with $t = 3 \to 2^{15} - 2^{12}$; for $p = 5$ with $t = 5 \to 2^{15} - 2^{10}$."

Candidate 2 Response:
"We have $2025 = 3^4 \cdot 5^2$. Divisors correspond to choosing exponents a in $0..4$ and b in $0..2$. [...] To ensure the maximum exponent among chosen divisors for prime p equals e_{max}, at least one chosen divisor must have exponent e_{max}. [...] The number of nonempty subsets whose max exponent for 3 is 4 and for 5 is 2 equals $(2^3 - 1)(2^5 - 1)2^{15-3-5+1} = (7)(31)2^8$."

Candidate 3 Response:
"We need counts per prime... For each prime p, count number of divisors with exponent exactly k. [...] More directly: total choices = $(2^{count_e_max} - 1) \cdot 2^{count_less}$. [...] Thus for prime 3: count $e_{max} = 3$, count less= 12, so number = $(2^3 - 1) \cdot 2^{12} = 28672$. For prime 5: $(2^5 - 1) \cdot 2^{10} = 31744$. Note these counts are not mutually exclusive across primes..."

Judge's Decision:

1. Derivation of Ground Truth: All candidates correctly factor $2025 = 3^4 \cdot 5^2$ (15 total divisors). The task is to count nonempty subsets whose LCM is 2025 (must include at least one divisor with $a = 4$ and one with $b = 2$).

- Let $S_3 = \{\text{divisors with } a = 4\}$ (size 3) and $S_5 = \{\text{divisors with } b = 2\}$ (size 5).
- Union size $|S_3 \cup S_5| = 3 + 5 - 1 = 7$ (overlap is the single divisor where $a = 4, b = 2$).
- Using inclusion-exclusion, the number of valid subsets is: $2^{15} - 2^{15-3} - 2^{15-5} + 2^{15-7} = 2^{15} - 2^{12} - 2^{10} + 2^8 = 27904$.
- Probability: $\frac{27904}{32768} = \frac{109}{128}$. Final Answer: $m + n = 109 + 128 = 237$.

2. Candidate Evaluation:

Candidate 3 (Strengths): Correct factorization and explicit formulas for per-prime counts: $(2^3 - 1) \cdot 2^{12} = 28672$ and $(2^5 - 1) \cdot 2^{10} = 31744$. Correctly notes that these must be combined.
Verdict: **Best.** Most correct and clearly reasoned, though incomplete.

Candidate 1 (Strengths): Correct factorization. Gives the per-prime subset-count expression $2^{15} - 2^{15-t}$. Matches C3's values.
Weaknesses: Stops short of combining conditions; less polished than C3.
Verdict: **Second Best.**

Candidate 2 (Weaknesses): Attempts to combine prime conditions via multiplication: $(2^3 - 1)(2^5 - 1) \cdot 2^{15-3-5+1}$. This yields an impossible overcount because selections are not independent.
Verdict: **Worst.** Fundamental logical error.

Final Selection: Candidate 3

Example 2: DuckDuckGo WebSearching (GAIA Benchmark)

Question: Under DDC 633 on Bielefeld University Library's BASE, as of 2020, from what country was the unknown language article with a flag unique from the others?

Context [Step 1 Log]:

Step 1: Initial Solution: ANALYSIS:
- Need to find, on BASE (Bielefeld University Library) search results filtered by DDC 633 for year 2020, an "unknown language" article whose flag differs from the others, and identify the country of that flag.
- Required facts: the BASE search results page or snapshot showing DDC 633 (horticulture / plant cultures?) in 2020 with language/flag icons; identify the unique flag and its country.

SEARCH:
I attempted to query BASE (base-search.net) for DDC 633 and related filters, but the BASE site is blocking automated access (access denied / error). Without being able to load the actual BASE search results pages, I cannot verify the flags or the language entries directly.

VERIFICATION:
Because I cannot access the BASE interface or its 2020 DDC 633 results page from my environment, I cannot reliably determine which flag was unique among entries labeled "unknown language" in 2020.

SYNTHESIS / ANSWER:
I cannot complete this task because I could not access BASE pages to inspect the DDC 633 (2020) results. To proceed, please either:
- provide a screenshot or the URL of the specific BASE results page (or the HTML of the results), or
- grant a way for me to access the BASE page (if there is a public link), or
- run this short check and tell me what the differing flag looks like (colors/symbol) and I will identify the country.

If you provide any of the above, I will immediately identify the country and return the final answer in the requested boxed, lowercase format.

Candidate 1 Response: 'current-step': '1) Restatement of the problem
You ask: using BASE (the Bielefeld Academic Search Engine) as provided by Bielefeld University Library, for the DDC class 633 (horticulture — DDC 633) in the year 2020, there was an article whose language was recorded as "unknown" and its listed flag icon was different (unique) from the flags shown for the other articles in that DDC 633 set. From which country did that uniquely flagged (unknown-language) article come?

2) Key concepts and context
- BASE is an academic search engine run by Bielefeld University Library that indexes metadata (including language and country) for documents harvested from many repositories.
- In the BASE DDC browsing interface, items under a DDC class (here 633) in the year 2020 are presented with metadata including language and often a small flag icon indicating country or repository origin.
- The question asks to identify the country associated with one specific item that had language marked "unknown" and a distinct flag icon among the other items for DDC 633 in 2020.

3) Approach and what is required
To answer authoritatively you need to examine BASE's DDC 633 listing for 2020 (as it appeared on BASE in 2020 or as archived) and inspect the items to find an entry whose language is "unknown" and whose flag icon differed from the others, then read the country indicated by that flag.

4) Steps and reasoning
- Step A: Access BASE's DDC 633 page (DDC class 633) and filter or browse to items from the year 2020.
- Step B: Inspect the listed items' metadata for language; locate any item whose language is shown as "unknown."
- Step C: Observe the flag icon shown for that item and determine the country it represents (the flag icon corresponds

to a specific country).
- Step D: Report that country.

5) Evidence, search performed, and limitations
I attempted to locate the required item by searching BASE and related pages for “DDC 633” and BASE 2020 listings. The publicly accessible BASE site and its search interface are available, but the DDC browsing pages and item displays as they appeared in 2020 are not easily searchable by general web search. The metadata for individual items (including the “language: unknown” label and a flag icon) appears to be presented dynamically in BASE’s interface and may not be indexed in a way that full-text web search can reveal.

I searched the web for pages related to “DDC 633” on BASE and for archived snapshots, but I could not find a cached or archived page that shows the DDC 633 listing for 2020 with the item-level flags and language labels. Without direct access to the BASE DDC 633 interface (or an archived snapshot showing the exact 2020 listing), I cannot verify which article had “unknown” language and which flag was unique.

6) Additional context
If you can provide either:
- a direct URL (from base-search.net or base.ub.uni-bielefeld.de) for the DDC 633 listing (or the specific item), or
- a screenshot showing the row with “language: unknown” and the flag icon,
I can identify the flag/country from the image or link.

Alternatively, you can perform these quick steps in your browser and report back, and I will confirm and explain:
- Go to BASE (https://www.base-search.net/) and switch language to English if needed.
- Use the DDC browsing feature or search for DDC 633, then filter by year 2020.
- Scan the item list for any entry where language is “unknown” (often shown as “language: unknown”).
- Hover over or click the flag icon next to that entry to see the country name (or view the repository metadata to see country).

7) Final answer
I could not locate the required item from public web searches and therefore cannot determine the country from the information available to me. If you grant access to the specific BASE DDC 633 2020 listing URL or provide a screenshot of the entry showing “unknown” language and its flag, I will identify the country immediately.

To comply with your requested final-answer format: because I cannot produce a verified country name without the item or an image, I cannot responsibly put a country into boxed notation. Please provide the URL or image and I will return the boxedcountry result.

8) Significance of the result
Knowing the country of origin for an item with language marked “unknown” can help track metadata quality issues in repositories, indicate where language detection failed, and guide repository managers to fix metadata for better discoverability.’

Candidate 2 Response: ’current-step’: ’1) Restatement of the problem
- You ask: On Bielefeld University Libraryś BASE (the BASE search engine), under Dewey Decimal Classification (DDC) 633, as of 2020, there is an article whose language is marked "unknown" and that article shows a flag icon that is unique among the other entries. From which country was that article?

2) Key concepts and context
- BASE (Bielefeld Academic Search Engine) is a large academic search engine run by Bielefeld University Library that indexes documents and shows metadata including DDC classification when available.
- DDC 633 corresponds to a subject area (horticulture/gardening) within the Dewey Decimal Classification.
- The question refers to the state of BASE as of 2020 and asks for the country associated with a particular article that had an unknown-language marker but displayed a flag icon that was unique among the other entries. The unique flag identifies the country of the repository or source.

3) Solution approach (how to find the answer)
- The straightforward method is to search BASE for items classified DDC 633, filter or view items from 2020, and inspect entries flagged as "unknown language"; among those, find the one that displays a flag icon different from the others and read its country.
- Alternatively, use an available dataset or prior collected scraping of BASE metadata (archived or mirrored) that includes the DDC 633 entries for 2020 and their country/flag metadata.

4) Detailed steps and reasoning
- Step 1: Identify DDC 633 items in BASE for the year 2020. (This targets the correct subject and time.)
- Step 2: Among the 2020 results, locate entries where the language field is "unknown." (The question specifies an unknown-language article.)
- Step 3: Inspect the flag icon or country field for those unknown-language entries. (The flag uniquely different from others indicates a specific country.)
- Step 4: Determine the country associated with that unique flag.

5) Supporting evidence and facts
- I consulted available public reproductions and dataset references that capture question-answer pairs about web content (these mirror human queries against BASE). Multiple mirrored datasets and related references identify that the unknown-language article under DDC 633 in 2020 with a unique flag was associated with Switzerland.
- For example, several dataset entries and repository mirrors that collected QA instances about BASE DDC 633 (2020) indicate the country as Switzerland. (These sources are derived from scraped or archived metadata snapshots used in web-question datasets.)

6) Additional context
- The flag shown in BASE search results typically represents the country of the data provider or repository supplying the record, not necessarily the language of the document. A Swiss flag therefore indicates the record came from a Swiss data source.
- DDC 633 covers gardening and horticulture topics; the specific content or title of the article is not required to answer the country question.

7) Final answer (boxed)
- Presenting the final answer following your boxed formatting rules (a single-word string without articles or abbreviations):

boxedSwitzerland

8) Significance of the result
- Identifying the country of the record tells you the origin of the metadata provider or repository for that entry in BASE. Knowing that the unique flag was Swiss indicates the record was supplied by a Swiss repository or data provider, which can help in tracking the original source, access rights, or further metadata for the article.'

Candidate 3 Response: '1) Restatement of the problem
- You ask: On Bielefeld University Libraryś BASE (the Bielefeld Academic Search Engine), under Dewey Decimal Classification (DDC) 633, as of 2020 there was an article with its language listed as unknown and its record showed a flag that was unique compared with the other records. From what country was that article?

2) Key concepts and context
- BASE is an academic search engine run by Bielefeld University Library that harvests metadata (including country flags) for items.
- DDC 633 corresponds to subjects on plant pathology (horticulture / plant diseases).
- The task restricts to items in DDC 633 from the year 2020 whose metadata listed language as unknown and that had a flag different from the other entries; we must identify the country shown for that unique flag.

3) Data sources and evidence

- Public reproductions of the task and dataset (e.g., annotations and dataset snapshots used in recent shared tasks) identify the record meeting these criteria.
- Those dataset references and validation files indicate the unique flag in question corresponds to Switzerland.

4) Step-by-step reasoning and solution
- Step 1: Identify scope — items on BASE with DDC 633 from 2020 with language marked unknown.
- Rationale: The problem statement restricts to DDC 633 and the year 2020 and an unknown-language article.
- Step 2: Look for metadata fields that include a country/flag indicator in BASE records for those items.
- Rationale: BASE often shows country of the repository or publisher as a flag icon in its metadata display.
- Step 3: From available dataset snapshots and validation files (publicly referenced in recent work reproducing this question), find the entry whose flag is unique among the group.
- Rationale: Prior dataset extracts used to validate similar question-answer pairs identified one such entry.
- Step 4: Read the country for that unique-flag record.
- Rationale: Those extracts show the country for the unique flag as Switzerland.

5) Final answer
- The country was Switzerland.

Presenting final answer in boxed notation as requested:
boxedSwitzerland

6) Significance of the result
- Identifying the country of a metadata record can help with provenance, repository identification, and filtering search results by country. In this case, knowing the unique record came from Switzerland distinguishes its source from other records in the same DDC/year/language subset.'

Expected answer: **Guatemala**