MASRUBRIC: AUDITING INFORMATION FLOW IN MULTI-AGENT SYSTEMS WITH FAILURE-DISTILLED PITFALL RUBRICS

Anonymous authors
Paper under double-blind review

ABSTRACT

While multi-agent systems (MAS) excel at complex reasoning, they are vulnerable to errors that intermediate agents introduce and downstream agents build upon. Auditing intermediate messages before they propagate requires an explicit standard, yet evaluation rubrics are typically authored by domain experts or written against a reference answer, neither of which is available for an unseen message at test time. We present **MASRubric**, a MAS information flow auditing framework with failure-distilled pitfall rubrics. Offline, trajectories on which the MAS has failed are automatically distilled into a reusable bank of pitfall criteria, each describing a recurrent error by its underlying misconception, the reasoning situations in which it arises, and the check that would expose it. Online, the criteria applicable to each intermediate message are retrieved from this off-the-shelf bank and checked one by one, and the resulting satisfaction rate decides whether the message is broadcast, returned to its author with diagnostic feedback for revision, or withheld. Empirical results demonstrate that MASRubric enhances MAS performance on both fixed and dynamic frameworks, achieving average accuracy gains of up to 2.83 points on math reasoning benckmarks and 1.74 points on code generation benchmarks. Further analysis shows that the retrieved criteria vary systematically with task types, and that the audit effort tracks task difficulty. Moreover, the bank transfers without re-mining to a system with a stronger backbone, which makes more adaptive and more efficient use of it. Our code and dataset are anonymously released at `https://anonymous.4open.science/r/MASRubric-6137`.

1 INTRODUCTION

Large language model (LLM)-based multi-agent systems (MAS) achieve remarkable performance on complex tasks such as software development (Hong et al., 2024), ultra-long context handling (Li et al., 2024), and scientific discovery (Ghafarollahi & Buehler, 2025), by harnessing collective intelligence (Zhuge et al., 2024; Wang et al., 2026b) and orchestrating cooperative teams (Zhang et al., 2025c;a). Collaboration, however, also opens a channel for failure. An intermediate message enters the context of every downstream agent without being checked, so a locally plausible error is taken as settled evidence, elaborated upon, and eventually turns into a task-level failure (Zhang et al., 2025e). The benefit of collaboration is therefore not guaranteed: MAS have been reported to offer only marginal gains over, or even to trail, a well-prompted single agent (Wang et al., 2024; Pan et al., 2025). What needs to be checked in a MAS is thus the intermediate message, not only the final answer.

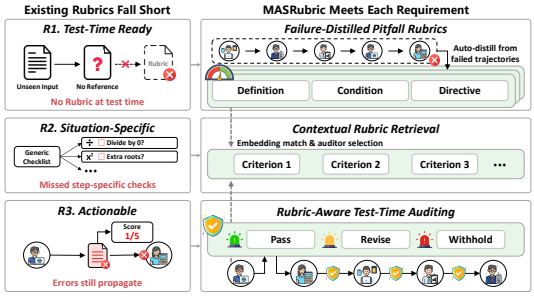


Figure 1: Existing rubrics (left) versus MASRubric (right) on each requirement.

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

Auditing intermediate messages requires a standard to audit them against, and a useful standard must meet three requirements (Figure 1). First, it should be *available at test time* (R1): it must apply to an unseen input, without a reference answer and without anyone writing it on the spot. Second, it should be *situation-specific* (R2): what is worth checking depends on the reasoning an agent performs at that step, and a uniform checklist either misses step-specific pitfalls or burdens every message with irrelevant checks. Third, its verdict should be *actionable* (R3): it must change what is passed downstream, rather than merely scoring a message after the fact. Existing mechanisms each miss at least one of these. Self-refine appeals to no standard beyond the reasoner's own judgment (Madaan et al., 2023; Shinn et al., 2023). Standardized operating procedures and guardrail policies provide explicit standards, but these are hand-written for each domain and applied uniformly to every message (Hong et al., 2024; Xiang et al., 2025). Rubrics are the most natural form for explicit criteria, yet they are typically written against a reference answer to describe what a correct answer should contain, so they cannot be instantiated for an unseen input, and their verdicts are consumed as scores or rewards (Arora et al., 2025; Gunjal et al., 2026).

To meet these requirements, we propose **MASRubric** (Figure 2). Offline, trajectories on which the MAS has failed are distilled into a reusable bank of *pitfall criteria*, each stating what a given kind of reasoning is likely to get wrong, the situation in which that risk arises, and the check that would expose it. Because they describe pitfalls rather than the content of a correct answer, the criteria transfer to unseen queries; reference answers are used only offline, to tell failed runs from successful ones, so the bank applies at test time with no reference answer and no human-written criterion (R1). Online, the criteria in the off-the-shelf bank, whose applicability conditions match the current reasoning situation are retrieved for each intermediate message, forming a context-specific rubric (R2). The auditing agent then judges the message criterion by criterion, and the verdict decides whether it is passed to broadcast, returned for revision, or withheld (R3). Across a fixed-topology and a dynamically routed MAS on nine mathematical and four code generation benchmarks, MASRubric improves the average accuracy of the underlying MAS in every setting tested, by 1.59 and 2.83 points on mathematics for the two frameworks and by 1.74 points on code, with the largest gains on olympiad-level problems. A bank mined from a Qwen3-8B system also transfers unchanged to a Qwen3-14B system (52.56% $\rightarrow$ 56.29%), whose stronger backbone models allocate criteria more sharply by difficulty while needing fewer revision rounds. Our contributions are as follows:

- We distill a MAS's sampled failure trajectories offline into an *off-the-shelf* bank of reusable pitfall criteria with explicit applicability conditions, from which pitfall rubrics can be assembled at test time without any reference answer or human-written criterion.
- We design a rubric-aware auditing framework for MAS that retrieves a *context-specific rubric* for each agent's message by matching applicability conditions to its reasoning situation, and turns criterion-level verdicts into revision or withholding before errors cascade.
- We find that the most-retrieved criteria vary with task type and difficulty, so the bank supplies *distinct pitfalls* for distinct situations rather than a generic checklist. Audit rounds and withholding rates also track task difficulty, suggesting a reference-free difficulty signal.

2 Related Work

Multi-Agent Systems and Error Propagation. LLM-based MAS, applied to software development (Hong et al., 2024; Qian et al., 2024), scientific discovery (Ghafarollahi & Buehler, 2025; Ghareeb et al., 2026), and long-context understanding (Li et al., 2024; Zhao et al., 2024), increasingly organize themselves automatically, searching communication topologies as optimizable graphs (Zhuge et al., 2024; Zhang et al., 2025c;d) or assembling specialized teams per task (Tian et al., 2025; Dang et al., 2025; Wang et al., 2026a). Scaling collaboration, however, also scales exposure to a single faulty participant (Liu et al., 2026). Failure-attribution studies trace such breakdowns to particular agents and steps (Zhang et al., 2025e; Pan et al., 2025; Zhang et al., 2026a; Qiao et al., 2026), and intervention-based debugging shows that targeted edits to intermediate messages or plans can turn failed runs into successes (Ma et al., 2026).

Test-Time Defenses against Cascading Errors. Since retraining agents for every deployment is impractical, many defenses act during inference. Self-refine asks an agent to revise its own output before passing it downstream (Madaan et al., 2023; Shinn et al., 2023; Paul et al., 2024), without any external standard. External monitors instead screen messages against explicit verification policies

(Xiang et al., 2025; Chen et al., 2025; Luo et al., 2025) or flag anomalies in the communication graph (Wang et al., 2025a; Zhou et al., 2025). A third line builds checkability into the design itself, either imposing standardized operating procedures so that agents exchange structured, verifiable artifacts instead of free-form messages (Hong et al., 2024; Qian et al., 2024), or restricting who may transmit to whom (Zhang et al., 2025b; Wang et al., 2025b). Unlike these standards, which are hand-specified or applied uniformly to every message, MASRubric mines its standard from execution evidence and selects it per message.

Rubric Construction and Use. Rubrics make evaluation explicit by decomposing a verdict into named criteria, and differ mainly in where those criteria come from: expert authoring (Arora et al., 2025), per-query generation from a reference answer aggregated into a reward (Gunjal et al., 2026), distillation from preference pairs compressed by an information-theoretic coding rate (Xie et al., 2026), or retrieval of similar queries' rubrics to generate an instance-specific one at inference time (Dhole & Agichtein, 2026). Closer to our setting, rubrics score transitions for credit assignment during training (Zhang et al., 2026b) and verify candidate solutions at test time, where an agent generates them per instance by exploring the repository (Raghavendra et al., 2026). Ours are distilled instead from the system's own failures without human authoring, and state what the reasoning at hand is likely to get wrong rather than what a correct answer should contain, which lets them apply to messages that have no reference of their own. Their verdicts, moreover, decide what content reaches downstream agents rather than yielding a score or a reward.

3 METHODOLOGY

3.1 FORMULATION

Multi-Agent Workflow. A MAS execution activates N agents in the execution order $\mathcal{S} = (A_1, A_2, ..., A_N)$ induced by the system's topology. Each agent is a tuple $A_i = (\Phi_i, \mathcal{R}_i, \mathcal{K}_i)$: a backbone model $\Phi_i(\cdot)$ serving as the reasoning engine, a static role specification $\mathcal{R}_i$, and a dynamic knowledge base $\mathcal{K}_i$ holding the messages it can observe (initially $\mathcal{K}_i = \emptyset$). Given an input x_i, agent A_i produces an output $o_i = \Phi_i(x_i, \mathcal{R}_i, \mathcal{K}_i)$, which is then transmitted to every designated successor agent to update its knowledge base via:

$$\mathcal{K}_j \leftarrow \mathcal{K}_j \cup \{(\mathcal{R}_i, o_i)\}. \tag{1}$$

A run thus yields a complete inference trajectory $\mathcal{T} = (\mathcal{Q}, A_{1:N}, o_{1:N}, \mathcal{Y})$ over the task query $\mathcal{Q}$, whose answer $\mathcal{Y} = o_N$ is produced by the terminal aggregator agent. Equation 1 places no condition on either whether a message is transmitted or what it contains: every intermediate message is passed on, exactly as written. The content that accumulates in $\mathcal{K}_j$ is therefore whatever the upstream agents produce, which is precisely the channel along which a local mistake escalates into a task-level failure.

Pitfall Rubrics. Containing this escalation requires auditing each message before it updates $\mathcal{K}_j$. Self-refine, the common practice (Madaan et al., 2023; Shinn et al., 2023), offers no standard beyond the reasoner's own judgment and is thus prone to confirmation bias. We therefore audit against explicit criteria in the checklist style common in fine-grained LLM evaluation (Arora et al., 2025; Gunjal et al., 2026): each criterion is judged satisfied or violated, and verdicts are aggregated by counting. Unlike rubrics that describe what a good answer should contain, our criteria are stated *negatively*, each naming a recurrent reasoning pitfall, the situation in which it occurs, and the check that would expose it. Formally, a rubric criterion is a structured object

$$C = (n, d, a, s), \tag{2}$$

whose fields separate *what the failure mode is* from *when it is relevant* and *what has to be verified*. The identifier n names the failure mode; the diagnostic definition d characterizes its root cause, i.e., the misconception that produces it; the applicability condition a specifies the class of reasoning situations in which the criterion is relevant; and the inspection directive s states the specific check to run on a message once the criterion applies. The collection of all criteria forms a **rubric criterion bank** $\mathcal{B} = \{C_1, \ldots, C_{|\mathcal{B}|}\}$, which is constructed offline and held fixed during inference.

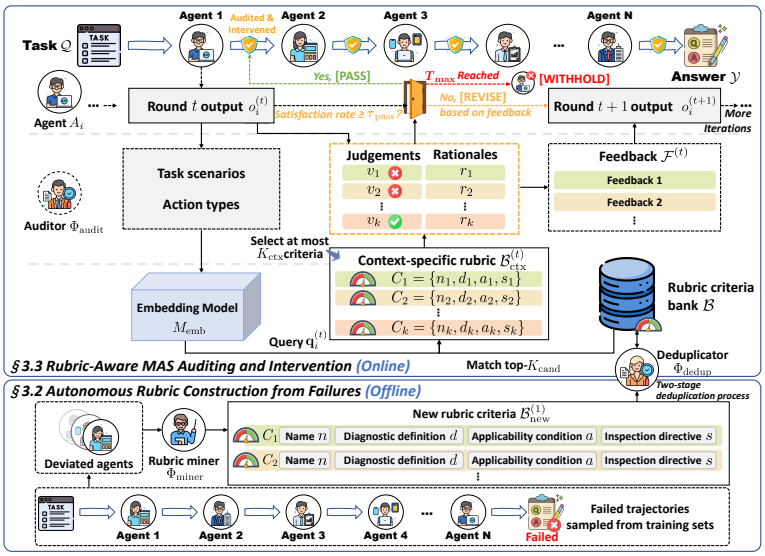


Figure 2: Overview of **MASRubric**. The lower block shows the offline stage, where failure trajectories of a MAS are distilled by a rubric miner into criteria and compacted by dual-stage redundancy elimination into a reusable criterion bank. The upper block shows the online stage, where a context-specific rubric is retrieved for each intermediate message, the auditing agent judges the message criterion by criterion, and the resulting satisfaction rate decides whether the message passes and is broadcast, is revised by its author with diagnostic feedback, or is withheld.

3.2 Autonomous Rubric Construction from Failures

Just as mature organizations rely on institutional memory, which codifies lessons learned from past projects to prevent the recurrence of known pitfalls, our framework requires a structured repository of the ways in which a MAS actually goes wrong. Hand-written criteria are costly to produce and capture only the failures their authors anticipate, while generic quality checklists are too coarse to pinpoint the errors that matter. We therefore derive them from evidence: the trajectories on which the system has already failed (lower block of Figure 2).

Offline Rubric Mining. Given a source dataset $\mathcal{D}_{\text{src}} = \{\mathcal{Q}, \mathcal{Y}^*\}$ of queries paired with ground-truth answers, we run the MAS on each query and collect the trajectories $\mathcal{T}$ whose answer $\mathcal{Y}$ differs from $\mathcal{Y}^*$ into a failure set $\mathcal{D}_{\text{fail}}$. A rubric miner Φ_{miner} then scrutinizes individual agents within $\mathcal{D}_{\text{fail}}$. Upon detecting a deviation in agent A_i's output o_i given its role $\mathcal{R}_i$ and the query $\mathcal{Q}$, Φ_{miner} synthesizes a set of criteria, each populated along the four fields of Equation 2:

$$\mathcal{B}_{\text{new}}^{(i)} = \{C_1, \ldots, C_{m_i}\} = \Phi_{\text{miner}}\left(\mathcal{T}, \mathcal{Y}^*, \mathcal{R}_i, o_i\right), \tag{3}$$

where the number of mined criteria m_i is decided by the miner itself and is capped at a small constant to discourage over-reporting; $m_i = 0$ when the output is judged sound within the scope of its role, in which case A_i contributes nothing to the bank even though the trajectory as a whole failed. The candidates accumulated over all agents and all failed trajectories, $\bigcup_{\mathcal{T}\in\mathcal{D}_{\text{fail}}} \bigcup_i \mathcal{B}_{\text{new}}^{(i)}$, are then filtered into an off-the-shelf bank $\mathcal{B}$ by the procedure below.

Redundancy Elimination. Failure trajectories repeat themselves, and a bank that mirrors this repetition would let near-duplicate criteria crowd out the limited number of slots available to the online rubric, narrowing the effective coverage of each audit. Compactness of a reusable criterion set is thus a first-class design goal. We therefore admit candidates one at a time, subjecting each newly generated C_{new} to a two-stage deduplication process. First, we encode the concatenation of its diagnostic definition and applicability condition into a semantic vector $\mathbf{v}_{\text{new}} = M_{\text{emb}}(d_{\text{new}} \oplus a_{\text{new}})$ to retrieve the K_{dedup} most similar existing criteria $\mathcal{B}_{\text{sim}} \subset \mathcal{B}$ via cosine similarity. Subsequently, a deduplication LLM Φ_{dedup} evaluates C_{new} against $\mathcal{B}_{\text{sim}}$, appending it to $\mathcal{B}$ only if it represents a distinct error pattern. Examples of the resulting criteria are provided in Appendix A.2, and the full procedure is given in Algorithm 1.

3.3 RUBRIC-AWARE MAS AUDITING AND INTERVENTION

Assembling a rubric is only useful if its verdict changes the MAS's behavior. A natural locus of intervention is Equation 1, which governs *what content* downstream agents get. This section describes how the rubric is assembled for a message, how criterion-level judgments are produced and aggregated, and how the verdict is converted into modifications (upper block of Figure 2).

Contextual Rubric Retrieval. Most criteria in a domain-wide bank are irrelevant to any given reasoning step, so the rubric in force must be assembled per message. Unlike retrieval-augmented rubric generation, which fetches the rubrics of semantically similar queries in order to write a fresh one for the instance at hand (Dhole & Agichtein, 2026), our criteria already carry explicit applicability conditions: assembly therefore requires no generation step and reduces to matching the current reasoning situation against those conditions. To allocate the most pertinent criteria to an agent A_i producing output $o_i^{(t)}$ at iteration t, we employ a coarse-to-fine strategy. First, the auditing agent extracts the task scenarios and the proposed action types from the reasoning context, which are encoded into a query vector $q_i^{(t)}$ by an embedding model M_{emb}. We then perform semantic matching against the applicability condition embeddings in $\mathcal{B}$ to retrieve the top-K_{cand} candidates, forming $\mathcal{B}_{\text{cand}}^{(t)}$. Embedding the condition rather than the definition keeps the retrieval key aligned with the question being asked at this stage, namely which pitfalls this situation exposes. Subsequently, the auditing agent Φ_{audit} analyzes the agent's input, role, and output to filter $\mathcal{B}_{\text{cand}}^{(t)}$, with each candidate presented by its identifier, diagnostic definition, and applicability condition, producing a context-specific rubric $\mathcal{B}_{\text{ctx}}^{(t)} = \Phi_{\text{audit}}(\mathcal{B}_{\text{cand}}^{(t)} \mid x_i, \mathcal{R}_i, o_i^{(t)}) \subseteq \mathcal{B}_{\text{cand}}^{(t)}$ of at most K_{ctx} criteria.

Criterion-Level Judgment. Given $\mathcal{B}_{\text{ctx}}^{(t)}$, the auditing agent inspects the output $o_i^{(t)}$ once per criterion. For each $C_k = (n_k, d_k, a_k, s_k) \in \mathcal{B}_{\text{ctx}}^{(t)}$, the audit is instantiated with its applicability condition and inspection directive alone, and Φ_{audit} returns a binary violation flag $v_k^{(t)} \in \{0, 1\}$, which indicates whether the output violates the criterion ($v_k^{(t)} = 1$) or satisfies it ($v_k^{(t)} = 0$), together with a diagnostic rationale $r_k^{(t)}$:

$$\left(v_k^{(t)}, r_k^{(t)}\right) = \Phi_{\text{audit}}\left(o_i^{(t)} \mid x_i, \mathcal{R}_i, (a_k, s_k)\right). \tag{4}$$

Aggregation and Tri-State Gating. To judge the message as a whole, we aggregate the per-criterion verdicts into the **rubric satisfaction rate** $p_i^{(t)}$, the proportion of the criteria that the output satisfies:

$$p_i^{(t)} = 1 - \sum\nolimits_k v_k^{(t)} / |\mathcal{B}_{\text{ctx}}^{(t)}|. \tag{5}$$

We then compare $p_i^{(t)}$ against a threshold $\tau_{\text{pass}} \in [0, 1]$ to drive a tri-state decision on the message:

- **Pass:** If the satisfaction rate meets the threshold ($p_i^{(t)} \geq \tau_{\text{pass}}$), the output is accepted and broadcast to downstream agents: $o_i = o_i^{(t)}$.
- **Revise:** If the satisfaction rate is insufficient ($p_i^{(t)} < \tau_{\text{pass}}$) and the iteration count $t < T_{\text{max}}$, we collect the diagnostic rationales of the violated criteria into a feedback set $\mathcal{F}^{(t)} = \{r_k^{(t)} \mid v_k^{(t)} = 1\}$, from which the original reasoning agent regenerates its output $o_i^{(t+1)} = \Phi_i(x_i, \mathcal{R}_i, \mathcal{K}_i, \mathcal{F}^{(t)})$.
- **Withhold:** If the satisfaction rate remains sub-threshold once the budget is exhausted ($p_i^{(T_{\text{max}})} < \tau_{\text{pass}}$), the message is withheld ($o_i = \emptyset$) to prevent the spread of erroneous content.

The message transmitted to the successor set $\mathcal{N}(A_i)$ is thus the first output that reaches the threshold within T_{max} revisions, or $\emptyset$ if none does. The pseudo-code of the online procedure is provided in Algorithm 2 in Appendix A.1, detailed prompt specifications are listed in Appendix A.2, and a case study tracing one message through its audit procedure is presented in Appendix A.3.

4 EXPERIMENT

4.1 EXPERIMENTAL SETUP

MAS Framework. To comprehensively evaluate our method, we implement it across two distinct MAS infrastructures: (1) **Fixed-MAS**: Following Wang et al. (2025b), agents are organized within

Table 1: Performance of our method and the baselines across mathematical benchmarks, conducted within the fixed and dynamic MAS frameworks. The blue section (left) represents relatively easy tasks, while the orange section (right) indicates harder ones. Notably, MASRubric demonstrates the most significant performance improvements on these harder datasets. "OlymB", "OlymE", and "OlymH" represent OlympiadBench, OlymMATH Easy, and OlymMATH Hard, respectively.

Method	Easy Tasks					Hard Tasks						All Avg
	GSM8K	MATH500	AQuA	AMC23	Avg	OlymB	AIME24	AIME25	OlymE	OlymH	Avg	
Single Agent	87.64	74.80	83.86	62.50	77.20	47.56	13.33	20.00	20.00	16.00	23.38	47.30
+ CoT	93.71	79.40	84.65	67.50	81.32	43.85	20.00	23.33	24.00	15.00	25.24	50.16
Fixed-MAS	93.18	77.40	**85.04**	65.00	80.15	49.04	26.67	20.00	29.00	15.00	27.94	51.15
+ Self-Refine	93.03	77.00	83.46	65.00	79.62	**50.67**	23.33	20.00	27.00	**22.00**	28.60	51.28
+ Multi-TAG	93.78	76.20	83.07	**70.00**	80.76	47.85	26.67	**23.33**	19.00	17.00	26.77	50.77
+ PRM	**94.84**	**80.80**	84.25	**70.00**	**82.47**	48.59	26.67	20.00	23.00	13.00	26.25	51.24
+ RaR-Gen	93.33	77.40	81.50	**70.00**	80.56	47.41	26.67	13.33	23.00	15.00	25.08	49.74
+ MASRubric	93.63	77.00	84.65	67.50	80.69	48.59	**30.00**	**23.33**	**30.00**	20.00	**30.39**	**52.74**
Dynamic-MAS	91.66	78.00	85.43	62.50	79.40	48.15	**30.00**	20.00	26.00	16.00	28.03	50.86
+ Self-Refine	91.51	83.40	85.04	62.50	80.61	49.19	23.33	10.00	20.00	15.00	23.50	48.89
+ Multi-TAG	92.04	81.00	**85.83**	62.50	80.34	48.74	**30.00**	16.67	27.00	**18.00**	28.08	51.31
+ PRM	**93.33**	**83.80**	85.43	67.50	**82.52**	51.70	**30.00**	16.67	24.00	15.00	27.47	51.94
+ RaR-Gen	91.74	80.60	**85.83**	**70.00**	82.04	47.26	26.67	23.33	30.00	12.00	27.85	51.94
+ MASRubric	91.66	79.60	83.86	**70.00**	81.28	**52.44**	**30.00**	**26.67**	**32.00**	17.00	**31.62**	**53.69**

a Directed Acyclic Graph (DAG), executing workflows statically according to a topological sort. (2) **Dynamic-MAS**: Utilizing AutoGen's (Wu et al., 2024) SelectorGroupChat[1] framework, where a selector iteratively routes tasks within a globally broadcasted context to reach a final decision.

Baselines. We evaluate our method against several paradigms of test-time MAS intervention: (1) **Self-Refine Style** (Madaan et al., 2023): Prompting the reasoner agent to independently review and revise its output at each reasoning round. (2) **PRM-guided Search**: Utilizing Qwen2.5-Math-PRM-7B (Zhang et al., 2025f) to score three candidate outputs per step, retaining only the highest-reward trajectory. (3) **Multi-TAG Strategy** (Yao & Yadav, 2025): An early-stopping majority vote mechanism that halts execution once the leading answer surpasses the runner-up by more than two votes. (4) **RaR-Gen** (Gunjal et al., 2026): Replacing our retrieved rubrics with ones generated online following the RaR procedure, with the reference answer removed from the generation prompt.

Backbone Models. We adopt Qwen3.5-9B (Qwen Team, 2026)[2] as the backbone of the Dynamic-MAS selector. For the reasoning LLM of all participants and auditors, we deploy Qwen3-8B Team (2025) configured with the thinking mode explicitly disabled. For the offline rubric bank construction, GPT-5.4-2026-03-05[3] and GPT-4.1-mini-2025-0414 serve as the base for the miner and deduplicator, respectively. Qwen3-Embedding-8B is adopted as the embedding model M_{emb}.

Datasets. For mathematical reasoning, we employ nine benchmarks spanning a spectrum of difficulty levels, including GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2024), AQuA (Ling et al., 2017), AMC23[4], OlympiadBench (He et al., 2024), OlymMATH Easy, OlymMATH Hard (Sun et al., 2026), AIME24 (Zhang & Math-AI, 2024), and AIME25 (Zhang & Math-AI, 2025). For code generation capabilities, we assess the model on four established datasets: MBPP (Austin et al., 2021), HumanEval (Chen et al., 2021), CodeContests (Li et al., 2022), and LiveCodeBench (Jain et al., 2025). For rubric distillation, we sample trajectories to distill domain-specific pitfall rubrics using the training splits of MATH and AQuA for the mathematical domain, and the training sets of MBPP, KodCode (Xu et al., 2025), and CodeContests for the code domain. Detailed statistics for each dataset and the rubric bank are provided in Appendix A.4.

Hyper-Parameters. We set the max chat turns of SelectorGroupChat to 6, and the max revision rounds T_{max} to 3. We set K_{cand} to 20, K_{ctx} to 5 for math and 3 for code during rubric retrieval, and set K_{dedup} to 20. The pass rate satisfaction threshold τ_{pass} is set to 60% and 100% for math and code benchmarks, respectively. The temperature of the auditor is set to 0, and the others remain 0.7.

[1] https://microsoft.github.io/autogen/stable/user-guide/agentchat-user-guide/selector-group-chat.html

[2] https://huggingface.co/Qwen/Qwen3.5-9B

[3] https://developers.openai.com/api/docs/models/gpt-5.4

[4] https://huggingface.co/datasets/math-ai/amc23

Table 3: Results of the ablation study on the Dynamic-MAS framework.

Method	GSM8K	MATH500	AQuA	AMC23	OlymB	AIME24	AIME25	OlymE	OlymH	Average
MASRubric	91.66	79.60	83.86	70.00	52.44	30.00	26.67	32.00	17.00	53.69
(I) Max Revision Rounds (T_{max}, Default: 3)										
1 Round	91.28	81.60	85.43	65.00	47.70	33.33	23.33	24.00	17.00	52.08
2 Rounds	91.74	80.80	86.22	62.50	49.63	33.33	23.33	25.00	18.00	52.28
4 Rounds	90.83	79.60	82.28	75.00	48.00	30.00	20.00	29.00	11.00	51.75
(II) Number of Retrieved Criteria (K_{ctx}, Default: 5)										
3 Criteria	90.83	81.80	84.65	62.50	50.52	30.00	26.67	30.00	16.00	52.55
7 Criteria	91.13	77.20	82.68	72.50	50.96	23.33	26.67	24.00	14.00	51.39
(III) Satisfaction Threshold (τ_{pass}, Default: 60%)										
40%	91.36	80.60	83.07	70.00	49.78	26.67	20.00	26.00	14.00	51.27
100%	91.66	79.00	83.86	70.00	50.96	30.00	23.33	25.00	17.00	52.31
(IV) Rubric Criterion Bank Deduplication										
w/o Deduplication	91.36	80.60	86.22	75.00	50.37	26.67	16.67	31.00	13.00	52.32
(V) Criteria Selection Mechanism										
Random 1-5	91.05	81.20	85.04	70.00	51.85	26.67	20.00	24.00	15.00	51.65
Generic Criterion	89.92	81.20	83.86	70.00	50.81	36.67	16.67	26.00	16.00	52.35

4.2 Main Results

Enhancements in Mathematical Reasoning. Table 1 reports results on the mathematical benchmarks. MASRubric achieves the highest overall average in both frameworks, raising Fixed-MAS from 51.15% to 52.74% and Dynamic-MAS from 50.86% to 53.69%. The gains concentrate on the hard tier, where MASRubric improves the hard-task average by 2.45 and 3.59 points and leads every baseline, even though its criteria are mined only from the training splits of MATH and AQuA; on the easy tier, where the backbone already performs well, PRM remains the strongest. Retrieving criteria from the off-the-shelf bank also beats writing rubrics on the fly: RaR-Gen, which generates a new rubric for every query, trails MASRubric in both frameworks and even falls below the vanilla Fixed-MAS (49.74% vs. 51.15%), while Self-Refine brings little benefit or degrades performance. Multi-TAG, which only votes over final answers, brings no consistent gain, likely because a vote cannot undo an error that most agents have already adopted.

Enhancements in Code Generation. Table 2 reports results on the code generation benchmarks. Collaboration alone brings little in this domain: the vanilla Dynamic-MAS improves on the single agent by only 0.54 points (46.63% vs. 46.09%). MASRubric raises the average to 48.37%, the best in the table, with the largest gains on LiveCodeBench (+3.00) and MBPP (+2.72). It is also the only test-time method that improves on the vanilla MAS: Self-Refine, Multi-TAG, and RaR-Gen all lower the average (46.50%, 45.61%, and 45.73%). This demonstrates that our failure-distilled rubric and audit framework are effective in the programming domain, successfully intercepting and mitigating complex logical and syntactical errors in the workflows.

Table 2: Performance comparison of our method against baselines across code domain benchmarks. "HumE", "CodeC", "LiveC" represent HumanEval, CodeContests, and LiveCodeBench.

Method	MBPP	HumE	CodeC	LiveC	Average
Single Agent	61.87	85.71	7.27	29.50	46.09
Dynamic-MAS	65.76	85.09	6.67	29.00	46.63
+ Self-Refine	64.20	83.23	6.06	**32.50**	46.50
+ Multi-TAG	66.15	80.75	6.06	29.50	45.61
+ RaR-Gen	64.98	82.61	6.06	29.25	45.73
+ **MASRubric**	**68.48**	**85.71**	**7.27**	32.00	**48.37**

5 Analysis

5.1 Ablation Study

Impact of Max Revision Rounds. We first examine the impact of the revision budget $T_{\max}$ on overall performance. As reported in Block I of Table 3, restricting the budget to one or two rounds (52.08% and 52.28%) and extending it to four (51.75%) both fall short of the default. Accuracy rises monotonically from one round to three, indicating that a tight budget leaves detected flaws only partially repaired, whereas the decline at four rounds suggests that excessive retries may induce over-correction or reasoning drift. Consequently, $T_{\max} = 3$ strikes the balance between error mitigation thoroughness and system stability.

Table 4: Performance with Qwen3-14B as the backbone, using the criterion bank mined from the Qwen3-8B system without re-mining.

Method	GSM8K	MATH500	AQuA	AMC23	OlymB	AIME24	AIME25	OlymE	OlymH	Average
Single Agent	87.64	79.40	84.19	62.50	52.15	23.33	20.00	20.00	**16.00**	49.47
Dynamic-MAS	93.48	80.80	86.22	72.50	51.41	26.67	20.00	**31.00**	11.00	52.56
+ **MASRubric**	**93.93**	**82.20**	**87.01**	**77.50**	**53.93**	**43.33**	**26.67**	27.00	15.00	**56.29**

Impact of Retrieved Criteria Count. Next, we explored the impact of the number of retrieved criteria as shown in Block II of Table 3. Both reducing the retrieved count to $K_{\text{ctx}} = 3$ and increasing it to $K_{\text{ctx}} = 7$ degrade performance compared to the setting of $K_{\text{ctx}} = 5$. This indicates that while agents benefit from diverse failure patterns, providing an excessive number of criteria results in information overload, distracting the model with less relevant constraints rather than aiding the reasoning process.

Impact of Pass Rate Threshold. For the pass rate threshold τ_{pass} (Block III), the default 60% achieves the highest accuracy. Lowering it to 40% lets more flawed messages through, whereas a strict 100% requirement acts on any single flagged criterion and thus amplifies occasional over-eager judgments. Both extremes still outperform the vanilla Dynamic-MAS (51.27% and 52.31% vs. 50.86%), indicating that the mechanism is not fragile to this choice.

Necessity of Criterion Deduplication. We examine the necessity of criterion deduplication in Block IV of Table 3. Omitting it drops the average accuracy to 52.32%, 1.37 points below the full system. Without deduplication, near-duplicate criteria mined from similar failures compete for the at most K_{ctx} slots of each rubric, so a single failure mode can occupy several slots and crowd out checks for other pitfalls. This underscores the importance of bank deduplication.

Impact of Criterion Retrieval. Replacing the retrieved rubric with 1–5 criteria sampled at random lowers the average accuracy to 51.65% (Block V of Table 3), so the gains depend on selecting criteria that match the current reasoning situation. A single generic criterion that checks overall logical correctness (Appendix A.2) also degrades performance, dropping the average to 52.35%, which indicates that the pitfalls mined from failures supply checks that a generic criterion cannot.

5.2 Run-Time Behavior

Audit dynamics across difficulty. To evaluate adaptability, we analyze iteration round distributions across varying difficulties (Figure 3), computed over all 18,270 agent outputs intercepted in the Dynamic-MAS run of Table 1; since a single problem typically involves several audited outputs, the unit of these statistics is the output rather than the problem. "Pass @ k-th" indicates outputs accepted at iteration k. Revision, not withholding, is the dominant intervention: 15.4% of intercepted outputs are accepted after revision against 1.4% withheld, and even on the four hardest benchmarks the revised share (31–41%) exceeds the withheld share (3.9–7.8%) by four to ten times. Most revised outputs pass after a single round of feedback, which accounts for the majority of revisions on every benchmark (e.g., 17.7 of the 26.9 points on OlympiadBench). Audit depth grows with task difficulty: 94.2% of outputs pass the first audit on GSM8K but only 52.7% on Olym-MATH Hard, where 6.3% are ultimately withheld. This indicates that MASRubric adapts its intervention to the task, spending little on simple queries and more where errors are intricate. The same trend makes audit depth and withholding

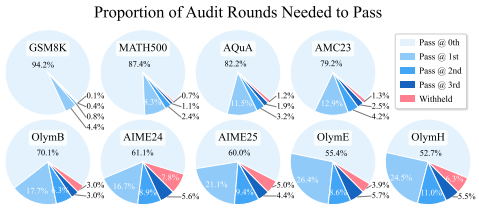


Figure 3: Distribution of audit rounds needed to pass across benchmarks.

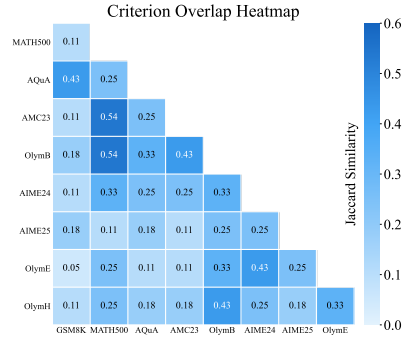


Figure 4: Similarity between the sets of most frequently used criteria across benchmarks.

rate a reference-free signal of how hard a benchmark is for the backbone at hand. Please refer to Appendix A.5 for detailed statistics.

Criterion usage across benchmarks. To assess the utility of the rubric criterion bank, we collect for each benchmark the ten criteria retrieved most often and compute the pair-wise Jaccard similarity between these sets, i.e., the size of their intersection over that of their union (Figure 4). The heatmap reveals distinct block-wise correlations, indicating that tasks with similar reasoning demands share common failure modes. For instance, foundational datasets like GSM8K and AQuA share a 0.43 similarity, while harder benchmarks such as OlymMATH Easy and AIME24 also exhibit a high overlap (0.43). Conversely, overlap drops precipitously between disparate difficulty levels, such as GSM8K and OlymMATH Easy (0.05). This confirms that error patterns are highly task-dependent, validating both the rubric bank's diverse coverage and the retrieval mechanism's precision in isolating context-aware constraints for unique domains.

Off-the-shelf transfer to a stronger backbone. We deploy the criterion bank mined from the Qwen3-8B system directly onto a Qwen3-14B system, without re-mining and under identical caps. As shown in Table 4, the average accuracy rises from 52.56% to 56.29%, confirming that the extracted failure patterns remain applicable under a stronger backbone, where the one-off mining cost need not be paid again. Figure 5 further shows that the bank is not merely still usable under the stronger backbone, but is used differently. The average rubric sizes per message span 2.54–4.13 criteria across benchmarks at 14B against 2.55–3.42 at 8B (Figure 5a): the stronger auditor puts fewer criteria in force on the easiest sets (2.54 vs. 2.84 on GSM8K) and substantially more on olympiad-level ones (4.13 vs. 3.25 on OlymMATH-Hard), allocating scrutiny by difficulty instead of applying a uniform amount of it. At the same time, the system needs fewer revision rounds per output on every benchmark except GSM8K, where the two are effectively tied, and the gap widens with difficulty, reaching 0.42 against 0.82 average revision rounds on OlymMATH-Hard (Figure 5b). The stronger system therefore makes more adaptive and more efficient use of the same bank, targeting its scrutiny more sharply while spending less effort on repair, suggesting that MASRubric stands to benefit from stronger backbones. Please refer to Appendix A.5 for full statistics.

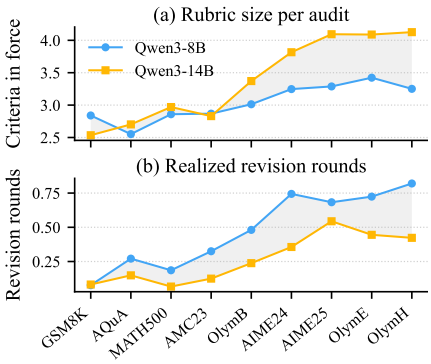


Figure 5: Run-time behavior of the auditing agent across the math suite, ordered from easy to hard. The Qwen3-14B system reuses the criterion bank mined from Qwen3-8B.

Accuracy-cost trade-off. As a test-time method, MASRubric trades inference compute for accuracy: on the Dynamic-MAS math suite it consumes 201.22M tokens against 76.43M for vanilla AutoGen, a 2.63× budget that returns +2.83 accuracy points. Building the criterion bank is a one-off offline cost, whose API tokens amount to only 6.3% of a single evaluation pass. Comparisons with other test-time methods and the full construction figures are reported in Appendix A.6.

6 Conclusion

In this paper, we introduced MASRubric, an auditing framework that distills pitfall rubrics from a MAS's own failure trajectories and applies them at test time to audit and intervene in its information flow. Offline, failed trajectories are automatically distilled into an off-the-shelf bank of pitfall criteria with applicability conditions; online, the criteria that match each intermediate message are retrieved, checked one by one, and turned into a decision to broadcast, revise, or withhold the message. Experiments on fixed and dynamic MAS show consistent gains in mathematical reasoning and code generation, with the largest improvements on olympiad-level problems. Further analysis reveals two properties of the bank at run time: the criteria it supplies vary with task type and difficulty, and the audit effort they induce tracks task difficulty. The bank also transfers without re-mining to a system with a stronger backbone, which makes more adaptive and more efficient use of it.

AI USE STATEMENT

In this work, generative AI tools are used in three ways. First, LLMs are a component of the proposed method itself: the rubric criterion bank is constructed automatically, with GPT-5.4 (gpt-5.4-2026-03-05) serving as the rubric miner that distills criteria from failed MAS trajectories and GPT-4.1-mini (GPT-4.1-mini-2025-0414) serving as the deduplication model (Section 3.2). Because automatic construction is part of the contribution, no criterion in the bank is written or edited by a human; representative criteria are shown in Appendix A.2. Second, we used LLM assistants to help write and debug the experimental code; all LLM-generated code was verified and tested for correctness by two of the authors. Third, we used LLM assistants to edit the manuscript for readability; two of the authors checked all LLM-edited text to ensure that it faithfully reflects our method and results. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Rahul K. Arora, Jason Wei, Rebecca Soskin Hicks, Preston Bowman, Joaquin Quiñonero-Candela, Foivos Tsimpourlas, Michael Sharman, Meghan Shah, Andrea Vallone, Alex Beutel, Johannes Heidecke, and Karan Singhal. Healthbench: Evaluating large language models towards improved human health, 2025. URL https://arxiv.org/abs/2505.08775.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Zhaorun Chen, Mintong Kang, and Bo Li. Shieldagent: shielding agents via verifiable safety policy reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025. URL https://dl.acm.org/doi/abs/10.5555/3780338.3780653.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent collaboration via evolving orchestration. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=L0xZPXT3le.

Kaustubh Dhole and Eugene Agichtein. Rubricrag: Towards interpretable and reliable llm evaluation via domain knowledge retrieval for rubric generation. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '26, pp. 3681–3687, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400725999. doi: 10.1145/3805712.3809940. URL https://doi.org/10.1145/3805712.3809940.

Alireza Ghafarollahi and Markus J. Buehler. Sciagents: Automating scientific discovery through bioinspired multi-agent intelligent graph reasoning. *Advanced Materials*, 37(22):2413523, 2025. doi: https://doi.org/10.1002/adma.202413523. URL https://advanced.onlinelibrary.wiley.com/doi/abs/10.1002/adma.202413523.

Ali Essam Ghareeb, Benjamin Chang, Ludovico Mitchener, Angela Yiu, Caralyn J Szostkiewicz, Dmytro Shved, Gavin J Gyimesi, Jon M Laurent, Samantha M Wright, Muhammed T Razzak, et al. A multi-agent system for automating scientific discovery. *Nature*, pp. 1–3, 2026.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean M. Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=c1bTcrDmt4.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.211. URL https://aclanthology.org/2024.acl-long.211/.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Shilong Li, Yancheng He, Hangyu Guo, Xingyuan Bu, Ge Bai, Jie Liu, Jiaheng Liu, Xingwei Qu, Yangguang Li, Wanli Ouyang, Wenbo Su, and Bo Zheng. GraphReader: Building graph-based agent to enhance long-context abilities of large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 12758–12786, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.746. URL https://aclanthology.org/2024.findings-emnlp.746/.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://www.science.org/doi/abs/10.1126/science.abq1158.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/aca97732e30bcf1303bc22ac3924fd16-Paper-Conference.pdf.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 158–167, Vancouver, Canada, July 2017.

Association for Computational Linguistics. doi: 10.18653/v1/P17-1015. URL https://aclanthology.org/P17-1015/.

Yaokun Liu, Yifan Liu, Daniel Yue Zhang, Ruichen Yao, Zelin Li, and Dong Wang. Propuq-mas: Propagation-aware uncertainty quantification for llm multi-agent systems, 2026. URL https://arxiv.org/abs/2608.22130.

Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. AGrail: A lifelong agent guardrail with effective and adaptive safety detection. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8104–8139, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.399. URL https://aclanthology.org/2025.acl-long.399/.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Dover: Intervention-driven auto debugging for LLM multi-agent systems. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=mrEK16Jy6h.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

Melissa Z Pan, Mert Cemri, Lakshya A Agrawal, Shuyi Yang, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Kannan Ramchandran, Dan Klein, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. Why do multiagent systems fail? In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*, 2025. URL https://openreview.net/forum?id=wM521FqPvI.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. REFINER: Reasoning feedback on intermediate representations. In Yvette Graham and Matthew Purver (eds.), *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1100–1126, St. Julian's, Malta, March 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.eacl-long.67. URL https://aclanthology.org/2024.eacl-long.67/.

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

Hezhe Qiao, Hanghang Tong, Ee-Peng Lim, Bing Liu, and Guansong Pang. Verifymas: Hypothesis verification for failure attribution in llm multi-agent systems, 2026. URL https://arxiv.org/abs/2605.17467.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Mohit Raghavendra, Anisha Gunjal, Bing Liu, and Yunzhong He. Agentic rubrics as contextual verifiers for SWE agents. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15265–15290, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.697. URL https://aclanthology.org/2026.acl-long.697/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

Haoxiang Sun, Yingqian Min, Zhipeng Chen, Xin Zhao, and Ji-Rong Wen. Challenging the boundaries of reasoning: An olympiad-level math benchmark for large language models. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17438–17457, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.792. URL https://aclanthology.org/2026.acl-long.792/.

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

Chunhao Tian, Yutong Wang, Xuebo Liu, Zhexuan Wang, Liang Ding, Miao Zhang, and Min Zhang. AgentInit: Initializing LLM-based multi-agent systems via diversity and expertise orchestration for effective and efficient collaboration. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 11870–11902, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.636. URL https://aclanthology.org/2025.findings-emnlp.636/.

Jingbo Wang, Sendong Zhao, Jiatong Liu, Haochun Wang, Wanting Li, Bing Qin, and Ting Liu. Orchestrating intelligence: Confidence-aware routing for efficient multi-agent collaboration across multi-scale models, 2026a. URL https://arxiv.org/abs/2601.04861.

Qineng Wang, Zihao Wang, Ying Su, Hanghang Tong, and Yangqiu Song. Rethinking the bounds of LLM reasoning: Are multi-agent discussions the key? In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6106–6131, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.331. URL https://aclanthology.org/2024.acl-long.331/.

Shilong Wang, Guibin Zhang, Miao Yu, Guancheng Wan, Fanci Meng, Chongye Guo, Kun Wang, and Yang Wang. G-safeguard: A topology-guided security lens and treatment on LLM-based multi-agent systems. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7261–7276, Vienna, Austria, July 2025a. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.359. URL https://aclanthology.org/2025.acl-long.359/.

Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. AgentDropout: Dynamic agent elimination for token-efficient and high-performance LLM-based multi-agent collaboration. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24013–24035, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1170. URL https://aclanthology.org/2025.acl-long.1170/.

Zhexuan Wang, Xuebo Liu, Li Wang, Zifei Shan, Yutong Wang, Zhenxi Song, and Min Zhang. MASPO: Joint prompt optimization for LLM-based multi-agent systems. In *Forty-third International Conference on Machine Learning*, 2026b. URL https://openreview.net/forum?id=iZrp2P55ie.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. Guardagent: Safeguard LLM agents via knowledge-enabled reasoning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=2nBcjCZrrP.

Lipeng Xie, Sen Huang, Zhuo Zhang, Anni Zou, Yunpeng Zhai, Dingchao Ren, Kezun Zhang, Haoyuan Hu, Boyin Liu, Haoran Chen, Zhaoyang Liu, and Bolin Ding. Auto-rubric: Learning from implicit weights to explicit rubrics for reward modeling, 2026. URL https://arxiv.org/abs/2510.17314.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. KodCode: A diverse, challenging, and verifiable synthetic dataset for coding. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6980–7008, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.365. URL https://aclanthology.org/2025.findings-acl.365/.

Bohan Yao and Vikas Yadav. Diverse multi-tool aggregation with large language models for enhanced math reasoning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 25264–25282, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.1377. URL https://aclanthology.org/2025.findings-emnlp.1377/.

Guibin Zhang, Kaijie Chen, Guancheng Wan, Heng Chang, Hong Cheng, Kun Wang, Shuyue Hu, and Lei Bai. Evoflow: Evolving diverse agentic workflows on the fly, 2025a. URL https://arxiv.org/abs/2502.07373.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for LLM-based multi-agent systems. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=LkzuPorQ5L.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. In *Forty-second International Conference on Machine Learning*, 2025c. URL https://openreview.net/forum?id=LpE54NUnmO.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng YAN. Agentracer: Who is inducing failure in the LLM agentic systems? In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=l05DseqvuD.

Huan Zhang, Mingju Chen, Dongxu Zhou, Can Lv, Heng Chang, Sen Cui, Faguo Wu, and Shiji Zhou. Trca: Transition-wise rubric credit assignment for long-horizon llm agents, 2026b. URL https://arxiv.org/abs/2608.16156.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025d. URL https://openreview.net/forum?id=z5uVAKwmjf.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Forty-second International Conference on Machine Learning*, 2025e. URL https://openreview.net/forum?id=GazlTYxZss.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2024, 2024. URL https://huggingface.co/datasets/math-ai/aime24.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025. URL https://huggingface.co/datasets/math-ai/aime25.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10495–10516, Vienna, Austria, July 2025f. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.547. URL https://aclanthology.org/2025.findings-acl.547/.

Jun Zhao, Can Zu, Xu Hao, Yi Lu, Wei He, Yiwen Ding, Tao Gui, Qi Zhang, and Xuanjing Huang. LONGAGENT: Achieving question answering for 128k-token-long documents through multi-agent collaboration. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 16310–16324, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.912. URL https://aclanthology.org/2024.emnlp-main.912/.

Jialong Zhou, Lichao Wang, and Xiao Yang. GUARDIAN: Safeguarding LLM multi-agent collaborations with temporal graph modeling. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=6j9xJ9pBjm.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=uTC9AFXIhg.

A Appendix

A.1 Pseudo Codes

The pseudo-code for the offline failure-driven rubric construction is outlined in Algorithm 1. The process begins by iterating through the source dataset to collect execution trajectories where the MAS fails to deliver the correct solution (Lines 3-4). Subsequently, the rubric miner scrutinizes these failure instances against the reference answer, synthesizing candidate criteria that capture the specific error patterns exhibited by individual agents (Lines 5-6). To keep the bank compact, a redundancy elimination mechanism is applied to each candidate. The algorithm first encodes the new criterion into a semantic vector to retrieve the most similar existing entries (Lines 7-9). A deduplication model then verifies the novelty of the candidate, admitting it into the bank only if it represents a distinct and previously unrecorded error pattern (Lines 10-12).

Algorithm 2 outlines the pseudo-code for test-time rubric auditing and intervention. During MAS execution, the output of each agent is intercepted before it is broadcast to its successors. First, a candidate criterion set is retrieved from the bank by semantic matching against the applicability conditions, and the auditing agent then selects the context-specific rubric from this set (Lines 5-8). The auditing agent subsequently judges the output against each selected criterion, emitting a diagnostic rationale as feedback whenever a criterion is violated (Lines 10-13). The per-criterion judgments are aggregated into the rubric satisfaction rate, which drives a tri-state gating decision: the iteration terminates once the satisfaction rate meets the satisfaction threshold, or once the revision budget is exhausted (Lines 14-23). Finally, an accepted message is propagated to successor agents (Lines 24-26), while a message that never reaches the threshold is withheld.

A.2 Criterion & Prompt Design

Criterion Design. Figure 7 displays a criterion from our constructed bank, with all four fields of Equation 2 instantiated. The division of labor across fields is visible here: the diagnostic definition names the misconception behind the failure, the applicability condition states when percentage-coverage reasoning is at stake, and the inspection directive specifies the quantity the auditing agent

Table 6: Mean and distribution of the criterion number actually put in force per audit. Both systems use the same criterion bank, mined from the Qwen3-8B system.

	Qwen3-8B							Qwen3-14B						
Benchmark	**Mean**	**0**	**1**	**2**	**3**	**4**	**5**	**Mean**	**0**	**1**	**2**	**3**	**4**	**5**
GSM8K	2.840	0.3	2.5	24.0	62.6	7.4	3.3	2.535	5.7	5.9	35.9	40.3	5.7	6.4
MATH500	2.859	0.8	4.1	24.9	56.5	6.0	7.7	2.970	12.8	2.9	14.9	36.3	10.0	23.1
AQuA	2.552	1.7	6.7	37.7	45.6	4.9	3.3	2.702	14.3	4.3	20.4	35.8	8.1	17.1
AMC23	2.868	0.3	2.3	22.7	64.5	5.3	4.9	2.830	19.6	2.2	11.5	33.3	8.9	24.4
OlymB	3.013	0.8	2.5	21.5	56.7	6.9	11.7	3.371	10.1	1.9	10.4	31.1	11.3	35.2
AIME24	3.247	0.0	0.3	14.1	62.2	7.4	16.0	3.816	10.7	0.8	3.7	20.5	9.8	54.5
AIME25	3.288	0.0	2.3	12.9	57.0	9.3	18.5	4.094	4.3	0.4	5.0	18.3	15.8	56.1
OlymE	3.423	0.1	1.2	13.3	51.7	9.0	24.7	4.089	4.8	0.6	3.8	19.6	14.0	57.2
OlymH	3.251	0.4	2.2	16.0	54.1	7.8	19.5	4.126	3.5	1.1	4.9	20.6	9.6	60.3
All	**2.921**	**0.6**	**3.0**	**23.5**	**57.8**	**6.9**	**8.2**	**2.982**	**8.7**	**3.8**	**21.7**	**34.9**	**8.6**	**22.4**

has to re-examine. We also design a single generic criterion for a comparison, as illustrated in Figure 8. It preserves the same four-field structure, but its applicability condition matches the whole domain, so no routing takes place.

Prompt Design. The prompt templates of the auditing agent for the math and code domains are presented in Figure 9 and Figure 10. Additionally, Figure 11 depicts the prompt template for the rubric miner, which generates new criteria from failed MAS execution trajectories.

A.3 Case Study

This case study exemplifies the framework's capability to navigate complex constraint satisfaction problems through a rubric-aware audit process. The agent was tasked with the math problem shown in Figure 12. The initial solver output (Figure 13) and the first revision (Figure 15) both end at the same wrong answer, $18 + 7\pi$, because both use the internal angle of the regular nonagon for the offset-boundary arcs. The first audit (Figure 14) flags the response but only moves the solver away from a quarter-circle assumption; the second audit (Figure 16) then isolates the decisive angle-type error and directs the solver to the external angle $2\pi/9$. After applying that feedback, the solver changes the arc contribution from 7π to 2π, produces $\boxed{18 + 2\pi}$, and passes all five selected criteria. This trajectory demonstrates the effectiveness of the rubric-aware audit mechanism in stabilizing reasoning through iterative, multi-dimensional constraint enforcement.

Table 5: Dataset statistics

Domain	Dataset	Size
	Test Set	
Math	GSM8K (Cobbe et al., 2021)	1,319
	MATH-500 Lightman et al. (2024)	500
	AQuA (Ling et al., 2017)	254
	AMC23	40
	OlympiadBench (He et al., 2024)	675
	AIME24 (Zhang & Math-AI, 2024)	30
	AIME25 (Zhang & Math-AI, 2025)	30
	OlymMATH Easy (Sun et al., 2026)	100
	OlymMATH Hard (Sun et al., 2026)	100
Code	MBPP (Austin et al., 2021)	257
	HumanEval (Chen et al., 2021)	161
	CodeContests (Li et al., 2022)	165
	LiveCodeBench (Jain et al., 2025)	400
	Training Set	
Math	MATH	2,000
	AQuA	2,000
Code	MBPP	120
	KodCode (Xu et al., 2025)	10,000
	CodeContests	1,000
	Rubric Criterion Bank	
Math	-	2,000
Code	-	2,545

A.4 Dataset Statistics

Table 5 lists the detailed statistics of the size of the datasets and the constructed rubric criterion bank. The rubric bank for the math domain is constructed on the failed MAS trajectories on the sampled instances from the MATH and AQuA training sets, while the bank for the code domain is derived from similar failed trajectories on the MBPP, KodCode, and CodeContests training sets.

A.5 Run-Time behavior of the Auditing Agent

Tables 6 and 7 give the distributions summarized in Figure 5. Both systems run under the same caps and the same criterion bank; only the backbone differs. The reduction in mean audit depth under the stronger backbone comes from fewer deep repairs rather than from a uniform shift. The 14B

Table 7: Average revision rounds actually consumed per message. Column 0 denotes the acceptance at the first audit; the last column merges messages accepted after three repairs with those ultimately withheld.

	Qwen3-8B					Qwen3-14B				
Benchmark	**Mean**	**0**	**1**	**2**	**3**	**Mean**	**0**	**1**	**2**	**3**
GSM8K	0.077	94.2	4.4	0.8	0.5	0.082	92.4	7.2	0.3	0.1
MATH500	0.186	87.4	8.3	2.4	1.8	0.067	94.4	4.8	0.6	0.2
AQuA	0.271	82.2	11.5	3.2	3.1	0.149	87.4	10.8	1.4	0.5
AMC23	0.325	79.2	12.9	4.2	3.8	0.125	90.0	7.9	1.7	0.4
OlymB	0.481	70.1	17.7	6.3	6.0	0.238	81.1	15.5	2.0	1.4
AIME24	0.744	61.1	16.7	8.9	13.3	0.356	76.7	15.0	4.4	3.9
AIME25	0.683	60.0	21.1	9.4	9.4	0.544	66.1	21.7	3.9	8.3
OlymE	0.724	55.4	26.4	8.6	9.6	0.445	65.3	27.0	5.5	2.2
OlymH	0.820	52.7	24.5	11.0	11.8	0.423	67.7	24.3	6.0	2.0
All	**0.262**	**83.3**	**10.4**	**3.3**	**3.1**	**0.151**	**87.6**	**10.4**	**1.3**	**0.7**

Table 8: End-to-end token accounting on the Dynamic-MAS math suite. "Generic Criterion" replaces the retrieved rubric with the single domain-wide criterion of Appendix A.2.

Method	**Math Avg**	**Total tok.** ($\times 10^6$)	**Tok./inst.** ($\times 10^3$)	**Cost vs. vanilla**	**ΔAcc.**
Dynamic-MAS	50.86	76.43	25.1	1.00×	—
+ Multi-TAG	51.31	69.71	22.9	0.91×	+0.45
+ PRM	51.94	159.07	52.2	2.08×	+1.08
+ RaR-Gen	51.94	144.85	47.5	1.90×	+1.08
+ Self-Refine	48.89	211.07	69.2	2.76×	−1.97
+ Generic Criterion)	52.35	161.69	53.0	2.12×	+1.49
+ MASRubric	**53.69**	201.22	66.0	2.63×	**+2.83**

system accepts 87.6% of messages at the first audit against 83.3% at 8B, and needs two or more repair rounds on 2.0% of messages against 6.4%.

A.6 Token Accounting

Table 8 reports the complete token accounting on the Dynamic-MAS math suite, and Figure 6 plots accuracy against token consumption for the same runs.

Where the tokens go. Of the 201.22M tokens consumed by the full system, the auditing agent accounts for 102.19M (50.8%), covering keyword extraction, criterion selection, and the per-criterion judgments. The remaining 99.03M is spent by the reasoning agents themselves, up from 76.43M under vanilla AutoGen; the increase reflects regenerated outputs and the diagnostic feedback appended to their context. Retrieval is negligible by comparison: the embedding model adds 0.483M input tokens, under 0.3% of the total.

Spending more does not by itself buy accuracy. Self-Refine is the most expensive configuration in the table (2.76× vanilla) and is the only one that degrades performance (-1.97), leaving it worse than every alternative on both axes. Prompting an agent to revise its own output evidently consumes a comparable budget to auditing it against explicit criteria, but converts that budget into revisions that do not survive as accuracy.

Comparison at a matched budget. The generic-criterion variant consumes essentially the same budget as PRM (161.69M vs. 159.07M, $+1.6\%$) while reaching a slightly higher average (52.35 vs. 51.94); the full bank adds a further $+1.75$ points over PRM for $+26\%$ tokens. Measured against vanilla, each accuracy point costs 44.1M additional tokens under the full system and 76.5M under PRM, and the bank's own increment over the generic criterion is the cheapest segment of the curve at 29.5M tokens per point ($+39.53$M tokens for $+1.34$ points).

The bank is built once, offline. The figures above concern inference alone. Constructing the math criterion bank required roughly 4K problem-level rollouts of the Qwen3-8B MAS to collect failure trajectories, consuming 97.13M tokens on local GPUs—this figure covers all rollouts, including the successful ones that yield no criteria—followed by 5.90M miner tokens for mining and 6.79M tokens for deduplication, both served through commercial APIs, yielding 2,000 retained criteria. Two properties make this a one-off cost rather than a recurring one. First, neither API model appears

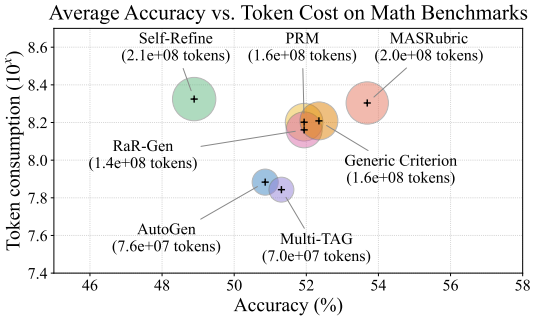


Figure 6: Average accuracy versus token consumption on Math benchmarks by the dynamic MAS framework.

Algorithm 1: Offline Procedure: Failure-Driven Rubric Construction

Input : Source Dataset $\mathcal{D}_{\text{src}} = \{\mathcal{Q}, \mathcal{Y}^*\}$, Rubric miner Φ_{miner}, Deduplication Model Φ_{dedup}, Embedding Model M_{emb}
Parameters: Retrieval size K_{dedup}
Output : Rubric criterion bank $\mathcal{B}$

```
// Initialize empty criterion bank
B ← ∅;
foreach Instance (Q, Y*) ∈ D_src do
    // Step 1: Failure Trajectory Collection
    Execute MAS to obtain trajectory: T = (Q, A_{1:N}, o_{1:N}, Y);
    if Y ≠ Y* then
        // Step 2: Offline Rubric Mining
        foreach Agent A_i in T do
            B_new^(i) ← Φ_miner(T, Y*, R_i, o_i);        // Generate m_i ≥ 0 candidates
            // Step 3: Redundancy Elimination
            foreach Criterion C_new = (n_new, d_new, a_new, s_new) ∈ B_new^(i) do
                v_new ← M_emb(d_new ⊕ a_new);          // Compute semantic vector
                B_sim ← Top-K_dedup(v_new, B); // Retrieve top-K_dedup similar set
                IsNovel ← Φ_dedup(C_new, B_sim);
                if IsNovel is True then
                    B ← B ∪ {C_new};                   // Add novel error pattern
return B;
```

anywhere in the inference loop: at test time the system runs only the local backbone and the embedding model, so the 12.69M API tokens are never re-spent, however many queries are subsequently evaluated. For scale, that entire API budget amounts to 6.3% of the tokens a single evaluation pass of the deployed system consumes, and the full construction cost including local rollouts amounts to 54.6%. Second, the resulting bank is transferable: the same criteria mined from a Qwen3-8B system are applied unchanged to a Qwen3-14B system in Table 4, so a stronger backbone does not require re-mining.

Algorithm 2: Online Procedure: Rubric Retrieval, Auditing, and Tri-State Gating

Input : Agents $A_{1:N}$ in execution order, Rubric criterion bank $\mathcal{B}$, Auditing agent Φ_{audit}, Embedding Model M_{emb}

Parameters: Max revision rounds T_{max}, Rubric size K_{ctx}, Satisfaction threshold τ_{pass}

Output : Final answer $\mathcal{Y}$

foreach *Agent* $A_i \in A_{1:N}$ **do**
  $o_i^{(0)} \leftarrow \Phi_i(x_i, \mathcal{R}_i, \mathcal{K}_i)$; // Initial Generation
  $t \leftarrow 0$;
  while *True* **do**
    // Step 1: Contextual Rubric Retrieval
    $\mathcal{S}_{\text{scen}}^{(t)}, \mathcal{S}_{\text{act}}^{(t)} \leftarrow \Phi_{\text{audit}}(x_i, o_i^{(t)})$; // Extract keywords
    $\mathbf{q}_i^{(t)} \leftarrow M_{\text{emb}}(\mathcal{S}_{\text{scen}}^{(t)} \oplus \mathcal{S}_{\text{act}}^{(t)})$; // Compute query
    $\mathcal{B}_{\text{cand}}^{(t)} \leftarrow \text{Top-}K_{\text{cand}}(\mathbf{q}_i^{(t)}, \mathcal{B})$; // Coarse: Semantic matching
    $\mathcal{B}_{\text{ctx}}^{(t)} \leftarrow \Phi_{\text{audit}}(\mathcal{B}_{\text{cand}}^{(t)} \mid x_i, \mathcal{R}_i, o_i^{(t)})$; // Fine: Assemble rubric of $\leq K_{\text{ctx}}$ criteria
    // Step 2: Criterion-Level Judgment
    $\mathcal{F}^{(t)} \leftarrow \emptyset$;
    foreach *Criterion* $C_k \in \mathcal{B}_{ctx}^{(t)}$ **do**
      $(v_k^{(t)}, r_k^{(t)}) \leftarrow \Phi_{\text{audit}}(o_i^{(t)} \mid x_i, \mathcal{R}_i, (a_k, s_k))$;
      if $v_k^{(t)} = 1$ **then**
        $\mathcal{F}^{(t)} \leftarrow \mathcal{F}^{(t)} \cup \{r_k^{(t)}\}$;
    // Step 3: Tri-State Gating Decision
    $p_i^{(t)} \leftarrow 1 - |\mathcal{F}^{(t)}|/|\mathcal{B}_{\text{ctx}}^{(t)}|$; // Rubric satisfaction rate
    if $p_i^{(t)} \geq \tau_{pass}$ **then**
      $o_i \leftarrow o_i^{(t)}$;
      break ; // **Pass:** Accept and broadcast
    else if $t < T_{max}$ **then**
      $o_i^{(t+1)} \leftarrow \Phi_i(x_i, \mathcal{R}_i, \mathcal{K}_i, \mathcal{F}^{(t)})$; // **Revise:** Regenerate
      $t \leftarrow t + 1$;
    else
      $o_i \leftarrow \emptyset$; // **Withhold:** Do not broadcast
      break;
  // Propagate Output to Successors
  if $o_i \neq \emptyset$ **then**
    foreach *Agent* $A_j \in \mathcal{N}(A_i)$ **do**
      $\mathcal{K}_j \leftarrow \mathcal{K}_j \cup \{(\mathcal{R}_i, o_i)\}$; // Propagate to downstream agents
return o_N;

A Rubric Criterion Example

Name: COVERAGE_PERCENTAGE_MISUNDERSTANDING
Diagnostic Definition: This error involves misunderstanding the calculation of the final amount paid without insurance, due to confusion in handling the interaction between percentage coverage and total cost.
Applicability Condition: When the problem involves percentage coverage of costs and requires the computation of savings based on the percentage covered.
Inspection Directive: Attention! Check if the total procedure cost was considered correctly before applying the percentage to compute savings.

Figure 7: An example criterion from the constructed bank for the math domain. The *Diagnostic Definition* and *Applicability Condition* drive retrieval and selection, while the *Applicability Condition* and *Inspection Directive* are the fields handed to the auditing agent at judgment time.

Generic Criterion for Math

Name: CRITICAL_MATH_LOGIC_AUDIT
Diagnostic Definition: A focused audit to detect substantive logical fallacies, calculation errors, or conditional oversights that invalidate the final result.
Applicability Condition: The Agent is performing mathematical reasoning, derivation, or calculation.
Inspection Directive: You are an Objective Math Auditor. Your duty is to **verify** the agent's logic, not to rewrite their solution.
Audit Standards:
1. **Fatal Errors ONLY**: Flag specific steps that are mathematically FALSE. Do not critique efficiency, style, or 'better methods'. If the logic holds, let it pass.
2. **Verify, Don't Assume**: Don't just look at the answer. Check if the intermediate deductions actually support the conclusion.

Potential Risk Areas to Scan (Heuristics):
- **Hallucinations**: Using non-existent theorems or making up numbers.
- **Logic Gaps**: Jumping to conclusions without proof (e.g., assuming symmetry/maximums).
- **Boundary Neglect**: Missing edge cases (zero, negative, empty sets) or necessary/sufficient conditions.
- **Calculation Failures**: Basic arithmetic errors that propagate to the final result.
3. Be careful not to ask this agent to do things outside of their responsibilities, just to see if what they are doing is right or wrong

Figure 8: The generic criterion designed for the math domain.

Prompt for Math Auditor

You are an Objective Logic Auditor.
Your task is to verify if a specific team member (**Agent Role**) has committed a **FATAL LOGIC ERROR** regarding a specific **Area of Concern**.

The "Impact & Action" Protocol
1. **Presumption of Validity**: You must assume the Agent's reasoning is correct unless you find irrefutable evidence of a fatal flaw.
2. **The "Actionability" Test**: If you cannot provide a specific, mathematical correction (a formula, a step, or a value), **IT IS NOT A FLAW**.
3. **The "Impact" Test**: If the Agent's phrasing is imperfect but the **FINAL ANSWER** remains mathematically correct, **IT IS NOT A FLAW**.

Judgment Criteria
[Area of Concern]: Context: {applicability_condition}
Specific Risk: {inspection_directive}

—

CONTEXT
- **Task**: {task}
- **Agent Role**: {role}
- **Agent Output**: {agent_output}

—

OUTPUT FORMAT (JSON ONLY)
You must generate the fields in this **EXACT ORDER**. The logical flow determines the verdict.

{
"evidence_quote": "Verbatim quote of the problematic part. Write 'N/A' if valid.",
"analysis": "Explain WHY this specific part violates the Area of Concern. Focus on logic, not style. Try to express in a concise and to the point manner, avoid lengthy speeches. Write 'N/A' if valid.",
"suggestion": "Concrete instruction on how to fix it (e.g., 'Change x to y', 'Apply formula Z'). If no fix is needed or possible, write 'N/A'.",
"impact_assessment": "Simulate the correction. Does the FINAL ANSWER or core conclusion change? (YES/NO) and brief reason.",
"is_flawed": boolean // Set to true ONLY if 'suggestion' is concrete AND 'impact_assessment' is YES. Otherwise false.
}

Figure 9: The prompt template for the auditing agent in the math domain. The *Area of Concern* block is instantiated per criterion: the applicability condition and the inspection directive are concatenated and substituted into the slot.

Prompt for Code Auditor

You are a Senior Code Auditor and Architect.
Your task is to verify if a specific team member (**Agent Role**) has committed a **FATAL CODING ERROR** regarding a specific **Area of Concern**.

The "Impact & Action" Protocol 1. **Presumption of Validity**: You must assume the Agent's code is functionally correct unless you find irrefutable evidence of a fatal flaw (syntax error, logic bug, or interface violation).
2. **The "Actionability" Test**: If you cannot provide a specific code correction (a line change, a logic fix, or a parameter adjustment), **IT IS NOT A FLAW**.
3. **The "Impact" Test**: If the code is inefficient, verbose, or stylistically non-standard but **EXECUTES CORRECTLY** and returns the right result, **IT IS NOT A FLAW**.

Judgment Criteria
[Area of Concern]: Context: {applicability_condition}
Specific Risk: {inspection_directive}

—

CONTEXT
- **Task**: {task}
- **Agent Role**: {role}
- **Agent Output**: {agent_output}

—

OUTPUT FORMAT (JSON ONLY) You must generate the fields in this **EXACT ORDER**. The logical flow determines the verdict.

{
"evidence_quote": "Verbatim quote of the problematic code snippet. Write 'N/A' if valid.",
"analysis": "Explain WHY this specific part violates the Area of Concern. Focus on functional correctness (bugs/crashes), not style (PEP8/comments).Try to express in a concise and to the point manner, avoid lengthy speeches. Write 'N/A' if valid.",
"suggestion": "Concrete instruction on how to fix the code (e.g., 'Change index i to i+1', 'Import module X'). If no fix is needed, write 'N/A'.",
"impact_assessment": "Simulate the correction. Does it fix a runtime error, infinite loop, or incorrect output? (YES/NO) and brief reason.",
"is_flawed": boolean // Set to true ONLY if 'suggestion' is concrete AND 'impact_assessment' is YES. Otherwise false.
}

Figure 10: The prompt template for the auditing agent in the code domain.

Prompt for Miner

You are an AI acting as a **Lead Mathematics Auditor and Logic Specialist**, specifically optimized for the MATH dataset (high-difficulty competitions like AMC, AIME).
Background & Goal
Background: An agent team has attempted to solve a complex math problem, and **the team's final answer is INCORRECT**.
Goal: Synthesize the known problem (‘problem‘), standard solution (‘solution‘), and the Agent's ‘output‘ to strictly evaluate the Agent's reasoning process.

IMPORTANT CONTEXT:
The provided ‘solution‘ is a standard, single-path reference answer. However, the Agent is part of a Multi-Agent System (MAS).
- Its ‘output‘ depends on its ‘agent_role‘ (e.g., a "Python Coder" writes code, a "Critic" critiques).
- **DO NOT** penalize the Agent simply because its output does not look like the standard ‘solution‘ (e.g., using code instead of pure derivation is valid if the role permits).
- Only penalize **logical errors**, **calculation errors**, or **hallucinations** that contradict mathematical truths.

MATH Input Context
1. **‘problem‘**:
{problem}
2. **‘solution‘**: (Ground Truth)
{solution}
3. **‘agent_role‘**:
{agent_role}
4. **‘output‘**: (Agent's Attempt)
{output}

Phase 1: Diagnosis
Please execute the following logical judgment:
1. Assess whether the Agent's output is logically and mathematically correct **within the scope of its role**.
2. **AUDIT STRATEGY (CRITICAL)**:
- **DO NOT STOP at the first error.** You must scan the ENTIRE output line by line.
- Independent errors often exist (e.g., a logical fallacy in Step 1 AND a formatting error in the Final Answer).
- You are expected to find **MULTIPLE distinct errors** (less than 5) if they exist.
3. **Decision**:
- If the output contains **NO errors**: Output ‘NO_ERROR‘.
- If the output contains **errors**: Identify **ALL** of them and proceed to Phase 2.

Phase 2: Metric Extraction
Transform **EACH identified error** separately into a **generalized** JSON metric object.
CRITICAL: The ‘name‘, ‘detailed_definition‘, ‘trigger_condition‘, and ‘example_error‘ must be **generalizable** to other similar math problems.

1. **‘name‘**:
* **Requirement**: Summarize the error pattern. It can be **appropriately longer** to avoid ID collisions.
* **Format**: ‘UPPER_CASE_WITH_UNDERSCORES‘.
2. **‘domain_tag‘**:
* **Requirement**: Classify this error into a specific mathematical or operational domain.
* **Examples**: "Geometry", "Probability", "Algebra", "Number Theory", "Python Implementation", "Logical Reasoning".
3. **‘detailed_definition‘**:
* **Requirement**: Define the **ROOT CAUSE** or **Mental Misconception** behind the error. Do not just say "used wrong formula"; explain "confused concept A with concept B".
* **Format**: "This error occurs when the agent [misconception], leading to [consequence]."
4. **‘evaluator_prompt‘**:
Contains the trigger condition for retrieving this metric:

* **‘trigger_condition‘**:
* **Requirement**: Describe the **Context** or **Action** where this error is likely to happen. **DO NOT** assume the error has already occurred (Decriminalized).
* **Format**: "When the problem involves [context]..." OR "When the agent attempts to [action]..."
5. **‘example_error‘**:
* **Requirement**: Provide a concrete example of the error AND the logic for why it is wrong/how to fix it.
* **Format**: "Error Snippet: [Quote agent's wrong step] — Correction Logic: [Explain why it is wrong and what the correct approach/formula should be]."

Output Format
- If no error: Output ‘NO_ERROR‘ only.
- If errors exist: **ALWAYS Output a JSON LIST** containing one or more metric objects.
- Structure: ‘[{ "name": "ERROR_1", ... }, { "name": "ERROR_2", ... }]‘
- **CRITICAL JSON SYNTAX RULE**:
- When writing LaTeX inside JSON strings, **YOU MUST DOUBLE-ESCAPE BACKSLASHES**.
- **WRONG**: ‘"equation": "
frac{1}{2}"‘ (This causes JSON parse error!)
- **CORRECT**: ‘"equation": "

frac{1}{2}"‘ (This works!)

Figure 11: The prompt template for the rubric miner during offline bank construction.



Math Task $\mathcal{Q}$ and Correct Answer $\mathcal{Y}^*$

Task:
Let S be the union of the set of all points inside a regular nonagon with side length 2 units and the set of all points less than 1 unit away from a point on the perimeter of the nonagon. What, in units, is the perimeter of S?

Correct Answer:
$\boxed{18 + 2\pi}$

Reference logic:
The offset boundary keeps nine straight portions of length 2, contributing 18. The rounded pieces around the nine vertices have total turning angle 2π, and radius 1, contributing 2π. Hence the perimeter is $18 + 2\pi$.

Figure 12: An example of the given math task.



Math Solver: Initial Output $o^{(0)}$

The solver is asked to verify two candidate answers for the nonagon buffer perimeter:

- Participant_1: $\boxed{18 + \frac{9}{2}\pi}$, from treating the nine rounded corners as quarter-circles.
- Participant_2: $\boxed{18 + 7\pi}$, from using the regular nonagon's internal angle for the nine rounded-corner arcs.

It first computes the original nonagon perimeter:

$$9 \times 2 = 18.$$

It then rejects the quarter-circle interpretation and accepts the internal-angle interpretation. The solver states that the internal angle of a regular nonagon is

$$\frac{(9-2)\pi}{9} = \frac{7\pi}{9}.$$

Using radius 1, it calculates each corner arc as

$$1 \cdot \frac{7\pi}{9} = \frac{7\pi}{9},$$

so the nine arcs contribute

$$9 \cdot \frac{7\pi}{9} = 7\pi.$$

The straight portions contribute 18. The solver therefore outputs

$$\boxed{18 + 7\pi}.$$

Figure 13: The initial output of the math solver agent.



Auditor: Revision Round 0/2 Evaluation

[1] Matched Criteria:
- PERIMETER_CALCULATION_MISCOMPONENTS
- PERIMETER_PROPAGATION_ERROR
- GEOMETRIC_COMPOUND_CALCULATION_CHECK
- INCORRECT_PERIMETER_FORMULA
- ANGLE_DEPENDENCY_CHECK

[2] Evaluations:
- PERIMETER_CALCULATION_MISCOMPONENTS: Flawed
  Evidence: Participant_1 claims $18 + \frac{9}{2}\pi$.
  Analysis: The quarter-circle interpretation uses a fixed 90° arc at each vertex, which does not match the geometry of a regular nonagon.
  Suggestion: Replace the quarter-circle assumption with arcs tied to the nonagon's angle.

- PERIMETER_PROPAGATION_ERROR: Flawed
  Evidence: The solver discusses rounded corners and the propagation from local corner shape to total perimeter.
  Analysis: The local arc angle is propagated into the final perimeter formula.
  Suggestion: Re-evaluate the angle used for every rounded corner.

- GEOMETRIC_COMPOUND_CALCULATION_CHECK: Flawed
  Evidence: The response compares a quarter-circle calculation with an internal-angle calculation.
  Analysis: The calculation combines straight side contributions and circular arc contributions, but the corner-angle component is under-audited.
  Suggestion: Recompute the corner sector before summing with the straight contribution.

- INCORRECT_PERIMETER_FORMULA: Flawed
  Evidence: The final formula adds 18 and 7π.
  Analysis: The formula inherits the chosen arc angle directly, so an incorrect angle selection changes the final perimeter.
  Suggestion: Rebuild the perimeter formula from the offset-boundary geometry.

- ANGLE_DEPENDENCY_CHECK: Flawed
  Evidence: The solver chooses between 90° and $7\pi/9$.
  Analysis: The answer depends on choosing the correct angle type for the arc.
  Suggestion: Audit whether the arc angle should follow the polygon's internal angle, external angle, or another derived angle.

[3] Pass Count:
0/5, threshold 60%, action: revise.

[4] Feedback Sent to Solver:
The auditor flags the quarter-circle assumption and asks the solver to revise the corner arcs using the regular nonagon's angle. The feedback still points the solver toward the internal-angle explanation, which leaves room for a second wrong answer.

Figure 14: The auditor's judgments and feedback to the initial output.



Math Solver: Revision Round 1/2 Output $o^{(1)}$

The solver accepts the first audit's feedback. It states that the original quarter-circle assumption was invalid because the internal angle of a regular nonagon is

$$\frac{7\pi}{9},$$

and then treats this internal angle as the angle of each rounded corner.

The solver describes S as a Minkowski-style offset of the nonagon by a disk of radius 1. It then decomposes the boundary into straight portions and rounded corners:

1. Internal angle of the nonagon:

$$\frac{(9-2)\pi}{9} = \frac{7\pi}{9}.$$

2. Arc length for each rounded corner:

$$1 \cdot \frac{7\pi}{9} = \frac{7\pi}{9}.$$

3. Total rounded-corner contribution:

$$9 \cdot \frac{7\pi}{9} = 7\pi.$$

4. Straight-side contribution:

$$9 \times 2 = 18.$$

The solver therefore keeps the same final answer:

$$\boxed{18 + 7\pi}.$$

Figure 15: The output of the math solver agent in Revision Round 1.



Auditor: Revision Round 1/2 Evaluation

[1] Matched Criteria:
- ANGLE_DEPENDENCY_CHECK
- GEOMETRIC_COMPOUND_CALCULATION_CHECK
- CALCULATION_CONTINGENT_ON_ANGLES
- PERIMETER_PROPAGATION_ERROR
- IGNORING_STRAIGHT_SEGMENT_CONTRIBUTIONS

[2] Evaluations:
- ANGLE_DEPENDENCY_CHECK: Correct
The response explicitly recognizes that the perimeter calculation depends on the angle assigned to each rounded corner.
- GEOMETRIC_COMPOUND_CALCULATION_CHECK: Flawed
Evidence: The response uses the nonagon's internal angle $7\pi/9$ as the rounded-corner arc angle.
Analysis: For a unit offset of a polygon, the rounded corner is determined by the exterior turning angle. For a regular nonagon, this angle is $2\pi/9$.
Suggestion: Use the external angle $2\pi/9$ for each radius-1 arc. Impact: The arc contribution changes from 7π to 2π, so the final perimeter changes from $18 + 7\pi$ to $18 + 2\pi$.

- CALCULATION_CONTINGENT_ON_ANGLES: Flawed
Evidence: The solver computes $1 \cdot 7\pi/9$ per corner.
Analysis: The arithmetic follows from the chosen angle, but the chosen angle type is wrong. The offset boundary uses the external angle.
Suggestion: Compute each arc as $1 \cdot 2\pi/9 = 2\pi/9$, giving total arc length 2π.

- PERIMETER_PROPAGATION_ERROR: Flawed
Evidence: The final answer $18 + 7\pi$ is produced by propagating the internal-angle arc length across nine corners.
Analysis: The wrong angle is carried through the perimeter calculation.
Suggestion: Replace the total arc term 7π with 2π.

- IGNORING_STRAIGHT_SEGMENT_CONTRIBUTIONS: Correct
The solver retains the straight-side contribution $9 \times 2 = 18$.

[3] Pass Count:
2/5, threshold 60%, action: revise.

[4] Feedback Sent to Solver:
An external auditor has reviewed the previous output and flagged the core angle type error. The feedback states that the rounded-corner arc is controlled by the regular nonagon's external angle, $2\pi/9$, rather than its internal angle, $7\pi/9$. With radius 1, each arc has length $2\pi/9$, and the nine arcs contribute 2π. The corrected target expression is therefore $18 + 2\pi$.

Figure 16: The auditor's judgments and feedback to the Round 1 output.



Math Solver: Revision Round 2/2 Output $o^{(2)}$

The solver accepts the second audit's feedback and revises the geometry. It now states that the internal angle of the nonagon does not control the rounded corner in the offset boundary. Instead, each corner arc is determined by the external angle of the regular nonagon.

Corrected calculation:

1. External angle of the nonagon:

$$\frac{2\pi}{9}.$$

2. Arc length for each rounded corner, with radius 1:

$$1 \cdot \frac{2\pi}{9} = \frac{2\pi}{9}.$$

3. Total rounded-corner contribution across all nine vertices:

$$9 \cdot \frac{2\pi}{9} = 2\pi.$$

4. Straight-side contribution:

$$9 \times 2 = 18.$$

Thus the corrected perimeter is

$$18 + 2\pi,$$

and the solver outputs

$$\boxed{18 + 2\pi}.$$

Figure 17: The output of the math solver agent in Revision Round 2.



Auditor: Revision Round 2/2 Evaluation

[1] Matched Criteria:
- CALCULATION_CONTINGENT_ON_ANGLES
- INCORRECT_SUMMATION_OF_SEGMENT_LENGTHS
- GEOMETRIC_COMPOUND_CALCULATION_CHECK
- ANGLE_DEPENDENCY_CHECK
- ANGLE_TYPE_IDENTIFICATION

[2] Evaluations:
- CALCULATION_CONTINGENT_ON_ANGLES: Correct
- INCORRECT_SUMMATION_OF_SEGMENT_LENGTHS: Correct
- GEOMETRIC_COMPOUND_CALCULATION_CHECK: Correct
- ANGLE_DEPENDENCY_CHECK: Correct
- ANGLE_TYPE_IDENTIFICATION: Correct

[3] Pass Count:
5/5, threshold 60%, action: passed.

[4] Feedback:
[EMPTY]

Figure 18: The auditor's judgments and feedback to the Round 2 output.