SELF-AGENT SYSTEM FOR REFLECTIVE REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reasoning reliability is critical as LLM agents become increasingly long-horizon, since local errors can silently accumulate and propagate across steps, yet the reliability of evolving reasoning trajectories remains underexplored. To address this, we introduce the **self-agent** perspective: an LLM-based agent treats its own evolving reasoning trajectory as the object of perception, deliberation, and action. Specifically, we realize this perspective by internalizing the agent loop over the model's own reasoning process. A tiny learned monitor acts as perception, identifying prefixes that are likely to derail later reasoning from past model-generated conversations and final-answer feedback. The frozen LLM itself then serves as deliberation, re-examining the suspect reasoning in a separate context rather than blindly continuing the flawed trajectory. A conservative commit gate provides action, writing a repair back only when it is specific and reliable. In this sense, self-agent turns a fixed LLM from a one-way generator into an online controller of its own reasoning trajectory, without retraining the generator, annotating intermediate steps, or relying on stronger planner/controller models or specialized expert agents. Comprehensive experiments on mainstream mathematical reasoning benchmarks show that Self-Agent provides plug-and-play gains across base models, improving long-horizon reasoning and recovering failing trajectories with only modest additional computation.

1 INTRODUCTION

The large language model (LLM) agent paradigm extends LLMs from single-turn response generation to goal-directed reasoning and action over multiple steps (Yao et al., 2022; Schick et al., 2023). Whether instantiated as a single autonomous agent with tools and memory (Packer et al., 2023) or as a multi-agent system with role-specialized components (Li et al., 2023; Wu et al., 2024), these systems share a common trait: the model produces a step-by-step reasoning trajectory in which each step is conditioned on all previous ones.

However, long-horizon, multi-turn agentic reasoning makes agent systems vulnerable to minor local errors that emerge along the trajectory, accumulate over time, and propagate to later decisions (Huang et al., 2023). Existing LLM-agent systems have primarily focused on expanding external capabilities (e.g., tools and memory) (Yao et al., 2022; Schick et al., 2023; Packer et al., 2023), training agent policies (e.g., fine-tuning or reinforcement learning) (Yang et al., 2025b), or coordinating multiple agents, leaving the reliability of evolving reasoning trajectories comparatively underexplored. From this trajectory-reliability perspective, three gaps become apparent. **(1) Unmonitored intermediate states.** Tools, memory, and task procedures expand the agent's external capabilities, but provide no mechanism for assessing whether its current reasoning state remains reliable (Lightman et al., 2023b; Yu et al., 2023; Chen et al., 2024). Once a mistaken belief enters the trajectory, it can become a flawed premise for later tool calls, memory updates, and continued generation, allowing a local error to cascade through the agent's subsequent decisions. **(2) Costly policy adaptation.** Training-based approaches can improve agent behaviors such as planning, tool use, and self-correction through supervised fine-tuning (SFT) or reinforcement learning (RL). However, they typically require additional training data, carefully designed environment-interaction trajectories, and updates to the generator itself (Yang et al., 2025b), making them less suitable when the base agent is fixed (e.g., closed-source LLM APIs) or costly to modify. **(3) Reliance on specialized role models.** Multi-agent systems typically improve complex reasoning by decomposing tasks among role-specialized agents, often using expert models to handle specific subtasks and a strong planner/controller model to allocate

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

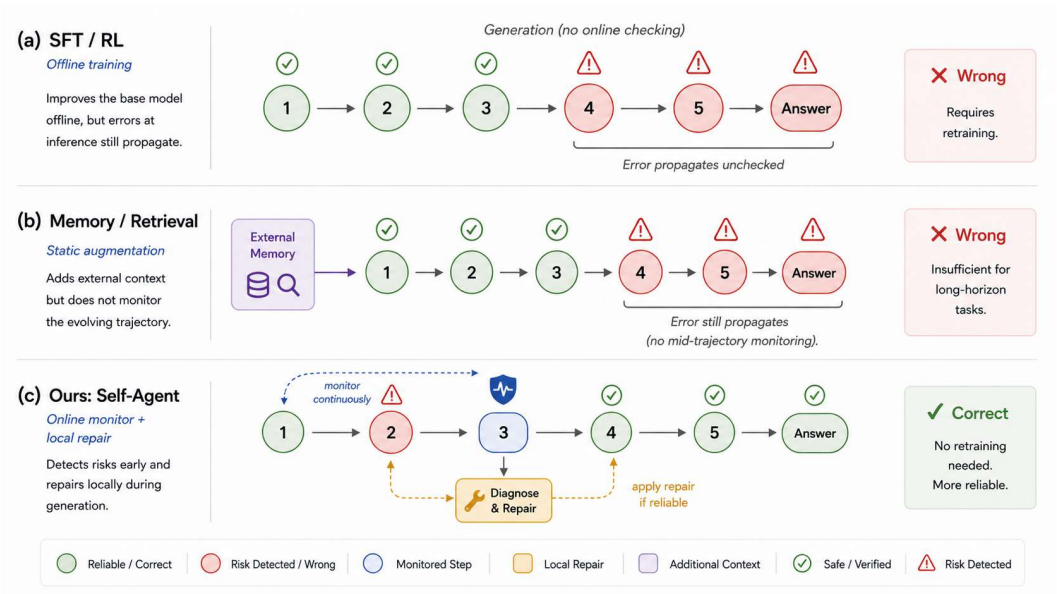


Figure 1: Motivation and method comparison. Top: training-based approaches such as SFT or RL can improve the base model offline, but they do not monitor whether a local error has already entered the current reasoning trajectory at inference time. Middle: memory or retrieval augments the model with additional external context, but still does not detect whether the ongoing trajectory has begun to derail. Bottom: Self-Agent instead monitors the evolving trajectory online, detects a risky step during generation, and invokes local diagnosis and repair before the error propagates to later steps.

work and integrate their outputs (Li et al., 2023; Wu et al., 2024). However, their effectiveness often depends on access to capable expert models and a strong planner/controller; if these role models are unavailable, costly to access, or unreliable, the performance of the entire system can degrade substantially. Motivated by this, we take a different perspective and ask: *can an agent improve the reliability of its own evolving reasoning trajectory while it is being generated?*

Inspired by this question, we introduce a **self-agent** perspective for trajectory-level reliability: *an LLM-based agent turns its own evolving reasoning trajectory into the object of perception, deliberation, and action*. Unlike post-hoc self-refinement methods that revise a completed output (Madaan et al., 2023) or self-evolving agents that update the model's policy across episodes (Shinn et al., 2023), and unlike direct self-correction settings where models often fail to revise flawed reasoning reliably (Huang et al., 2023), a self-agent internalizes an agentic control loop *within* a single ongoing generation: it observes risky prefixes, reflects on recent reasoning, decides whether intervention is needed, and commits a repair only when the correction is sufficiently reliable, and all without modifying the frozen generator.

Concretely, the self-agent system is built around three lightweight mechanisms. *First*, an online reflection monitor evaluates each reasoning prefix as it is generated and flags states whose continuation is likely to fail, preventing unreliable intermediate states from being blindly carried forward. *Second*, the reflection monitor is a tiny network (e.g., 2-layer MLP with 0.33M parameters), which can be updated online with minimal time and compute cost using past model-generated conversations and final-answer feedback, without annotations of intermediate reasoning steps. *Third*, when the monitor detects a high-risk prefix, the same fixed LLM is invoked in a separate diagnostic context to inspect the recent suspect steps and propose a local correction. A conservative commit gate then writes the correction back to the live trajectory only when it is concrete and reliable. In this way, our self-agent paradigm turns reflection into an online reliability mechanism for long-horizon reasoning, without requiring a stronger planner/controller or additional specialized models.

Our contributions are summarized as follows:

1. **A self-agent perspective for trajectory reliability.** We formalize the idea of treating an agent's own evolving reasoning trajectory as the object of perception, deliberation, and action. This perspective reframes online error correction as an agentic control problem over the model's own generation, complementary to stronger generators, additional tools, or multi-agent coordination.
2. **A lightweight, annotation-free reflection monitor.** We instantiate this perspective with a tiny learned probe (i.e., 0.33M parameters) over the frozen model's hidden states, trained from ordinary rollouts using only final-answer feedback. The monitor provides prefix-level risk signals without step-level human annotation or updates to the base generator.
3. **Empirical validation of online trajectory repair.** Experiments across four Qwen3 model scales on GSM8K, MATH-500, and AIME demonstrate effective trajectory repair with modest inference overhead; additional results on MBPP with Qwen3-4B and Qwen3-8B show that the rescue–harm trade-off also remains favorable for code generation.

2 Related Work

LLM agents and capability expansion. LLM-agent systems extend language models beyond single-turn response generation through tools, memory, actions, and multi-agent coordination. ReAct interleaves reasoning traces with task-specific actions, Toolformer studies tool invocation, and MemGPT augments agents with explicit memory management (Yao et al., 2023b; Schick et al., 2023; Packer et al., 2023). Multi-agent frameworks such as HuggingGPT, CAMEL, AutoGen, and MetaGPT decompose complex tasks across external models or role-specialized agents (Shen et al., 2023; Li et al., 2023; Wu et al., 2023; Hong et al., 2024). These works expand what agents can access, remember, or coordinate, but capability expansion alone does not ensure that the agent's evolving reasoning state remains reliable before it is reused by later actions, memory updates, or generation steps.

Inference-time reflection and trajectory verification. Several methods use reflection, critique, or verification to improve reasoning at inference time. Self-Refine revises outputs through iterative self-feedback, Reflexion uses verbal feedback and memory across attempts, and ReVISE learns intrinsic self-verification for test-time refinement (Madaan et al., 2023; Shinn et al., 2023; Lee et al., 2025). Meanwhile, verifiers, process supervision, and process reward models evaluate intermediate reasoning states for selection, search, reranking, or reinforcement learning (Cobbe et al., 2021; Lightman et al., 2023a; Yu et al., 2023; Chen et al., 2024; Setlur et al., 2025). Search-based methods such as self-consistency, Tree of Thoughts, Graph of Thoughts, and Language Agent Tree Search further exploit multiple candidate trajectories (Wang et al., 2023; Yao et al., 2023a; Besta et al., 2024; Zhou et al., 2024). These directions show that intermediate states contain useful reliability signals, but they typically use such signals for post-hoc revision, candidate selection, or explicit search, rather than for lightweight online intervention within one unfolding trajectory.

Training-based agent adaptation and self-improvement. Another line of work improves reasoning by updating model parameters or policies. STaR bootstraps reasoning data from successful rationales, V-STaR combines self-taught reasoning with learned verifiers, and Self-Rewarding Language Models use model-generated rewards for iterative training (Zelikman et al., 2022; Hosseini et al., 2024; Yuan et al., 2024). Agent-specific training pipelines such as SWE-smith further synthesize interaction trajectories to improve agent behavior in realistic environments (Yang et al., 2025a). These methods can produce stronger future behavior, but they require additional data, training cycles, environment interactions, or updates to the generator itself. Thus, reliability is mainly addressed through offline adaptation, leaving comparatively less explored how a fixed model's current reasoning trajectory can be monitored and corrected during inference.

3 Method

In this section, we first formulate online self-correction as a thresholded intervention problem (Sec. 3.1). We then introduce the overall Self-Agent structure (Sec. 3.2) and describe its three modules: the reflection monitor for prefix risk detection (Sec. 3.3), the diagnostic module for local error analysis (Sec. 3.4), and the commit gate for conservative continuation (Sec. 3.5).

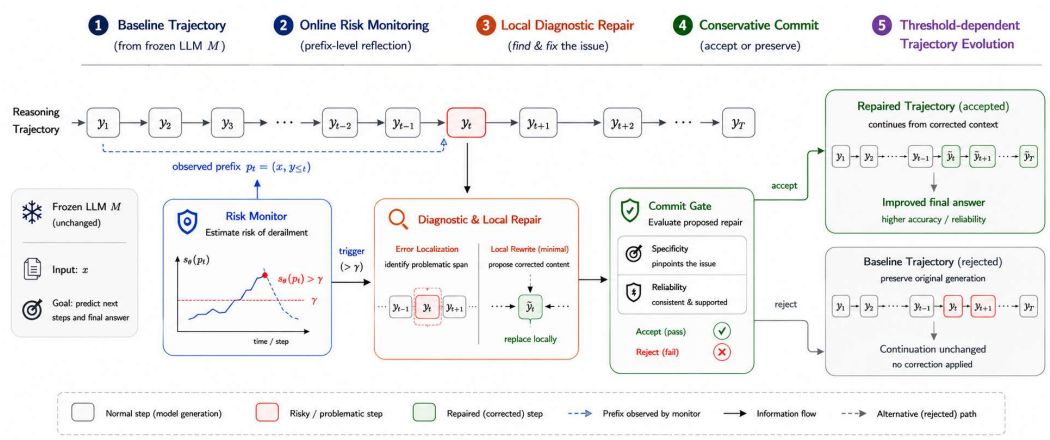


Figure 2: Overview of Self-Agent. The frozen language model first produces a baseline reasoning trajectory. A lightweight reflection monitor observes each prefix online and triggers intervention only when the risk score exceeds the threshold γ. The same frozen model is then reused in a separate diagnostic context to localize the suspect span and propose a minimal local repair. A conservative commit gate accepts only specific and reliable corrections; if accepted, generation resumes from the repaired state, and otherwise the original trajectory continues unchanged.

3.1 Problem Definition

Let M be a frozen language model acting as the agent. Given an input problem x, the model generates a reasoning trajectory $y = (y_1, \ldots, y_T)$, where each y_t denotes a reasoning step or action-like textual segment. We denote the prefix at step t as $p_t = (x, y_{\le t})$.

During generation, the system observes prefixes sequentially and must decide online whether to continue generation unchanged or invoke intervention. We formalize this as a thresholded intervention policy:

$$\pi_{\theta,\gamma}(p_t) = \mathbb{I}[s_\theta(p_t) > \gamma]. \tag{1}$$

where $s_\theta(p_t) \in [0, 1]$ is a learned risk estimator and γ is the intervention threshold.

When intervention is triggered, the system may perform diagnosis and local correction, producing a repaired trajectory $\tilde{y}_i^\gamma$. If no correction is accepted, the trajectory remains unchanged, i.e., $\tilde{y}_i^\gamma = y_i$.

We assume access only to outcome-level supervision through a task-level verifier $v(x, y) \in \{0, 1\}$, which evaluates whether a completed trajectory reaches the correct final answer.

The intervention threshold controls the trade-off between rescuing incorrect trajectories and unnecessarily modifying correct ones. Rather than optimizing γ through gradient updates, we select it from a finite candidate set Γ using validation performance:

$$\gamma^\star = \arg\max_{\gamma \in \Gamma} \sum_{i \in \mathcal{V}} \left[v(x_i, \tilde{y}_i^\gamma) - v(x_i, y_i) \right]. \tag{2}$$

Each positive term corresponds to a rescued trajectory, while each negative term corresponds to a harmed trajectory. This objective explicitly captures the selective-intervention trade-off between recovery and over-correction, and can additionally be constrained by bounded intervention budgets such as diagnostic calls, generated tokens, or wall-clock latency.

3.2 Self-Agent Overall Structure

Many long-horizon reasoning failures arise from local mistakes that enter the trajectory and are subsequently propagated by later reasoning steps. Self-Agent addresses this problem through selective online intervention: instead of applying reflection uniformly to every reasoning step, the system invokes additional reasoning only when the current trajectory appears high-risk.

Self-Agent consists of three modules that instantiate the intervention policy defined in Sec. 3.1. Given a reasoning prefix $p_t = (x, y_{\leq t})$, the **reflection monitor** s_θ predicts whether continuing from the current state is likely to fail. When the predicted risk exceeds the intervention threshold γ, the **diagnostic module** reuses the same frozen model M in a separate context to inspect the recent reasoning region and propose a localized correction. The **commit gate** then determines whether the proposed correction should modify the live trajectory. If accepted, generation resumes from the repaired context; otherwise, the original trajectory continues unchanged.

3.3 Reflection Monitor

The reflection monitor identifies high-risk reasoning prefixes without requiring expensive step-wise verification by a large language model. A simple alternative is to rely on local uncertainty features such as entropy or logit margin. However, these signals mainly capture local decoding ambiguity rather than downstream reasoning reliability: a trajectory can confidently continue along an incorrect reasoning path. Instead of estimating local token uncertainty, Self-Agent estimates prefix-level continuation risk.

For a reasoning prefix p, we define the continuation reliability:

$$Q(p) = \Pr_{y \sim M(\cdot|p)}[v(x, y) = 1], \tag{3}$$

which measures the probability that the frozen model reaches a correct final answer when continuing generation from p. Prefixes with low $Q(p)$ are treated as high-risk states because the model is unlikely to recover if generation proceeds unchanged. In practice, $Q(p)$ is estimated from a small number of sampled continuations from the frozen generator.

We implement the monitor as a lightweight probe over frozen reasoning representations. For each prefix p, we extract the last hidden state $h_M(p)$ of the frozen generator and parameterize the risk estimator as:

$$s_\theta(p) = \sigma(d_\theta(h_M(p))). \tag{4}$$

Only the lightweight probe d_θ is updated during training, while the generator M remains frozen.

Rather than regressing exact continuation reliability, we convert $Q(p)$ into thresholded binary supervision by treating low-$Q(p)$ prefixes as high-risk examples and high-$Q(p)$ prefixes as low-risk examples. At inference time, $s_\theta(p_t)$ is compared against the intervention threshold γ to determine whether the current reasoning prefix should trigger diagnosis.

3.4 Diagnostic Module

The diagnostic module proposes localized corrections once the reflection monitor identifies a prefix as high-risk. A direct alternative is to append a self-check prompt to the current trajectory and ask the model to reconsider its reasoning in place. However, autoregressive conditioning tends to reinforce the existing reasoning trajectory, making in-context self-checking unreliable after the model has already committed to an incorrect path (Huang et al., 2023). As shown later in Sec. B.3, direct self-checking rescues some failures but also introduces substantially more harmed trajectories.

Self-Agent therefore performs diagnosis in a separate context while still reusing the same frozen model M. When the monitor marks a prefix as high-risk, the system constructs a diagnostic context from the original problem, earlier reasoning history, and a recent suspect reasoning window surrounding the detected prefix. The frozen model M is then asked to inspect only this local region and produce a structured correction proposal:

$$z_t = (\delta_t, c_t, r_t, u_t), \tag{5}$$

where δ_t is the diagnostic decision, c_t is the confidence score, r_t is the proposed corrected fact, and u_t specifies the rewrite scope. The diagnostic output does not directly modify the live trajectory. Instead, it summarizes a candidate local intervention that is later evaluated by the commit gate.

3.5 Commit Gate

The commit gate determines whether a proposed correction should modify the live reasoning trajectory. Formally, it maps the diagnostic proposal z_t to a binary decision:

$$g(z_t) \in \{0, 1\}, \tag{6}$$

where $g(z_t) = 1$ indicates that the proposed correction is accepted and $g(z_t) = 0$ indicates rejected.

The gate is intentionally conservative. A correction is accepted only when the diagnostic module reports a high-confidence localized error, provides a non-empty corrected fact, and restricts the rewrite scope to a small local region. Otherwise, the proposal is discarded.

Given the gate decision, the updated reasoning prefix is:

$$\tilde{p}_t = \begin{cases} p_t, & g(z_t) = 0, \\ \mathrm{Repair}(p_t, r_t, u_t), & g(z_t) = 1. \end{cases} \tag{7}$$

The repaired prefix $\tilde{p}_t$ is used as the context for subsequent generation by the same frozen model M.

This separation between diagnosis and commitment makes intervention explicit and controllable: the diagnostic module proposes candidate local corrections, while the commit gate determines whether those corrections should affect the ongoing reasoning trajectory.

4 Experiments

4.1 Experimental Setup

Data. For the mathematical reasoning experiments, the training data for the **Reflection Monitor** comes from rollouts generated by Qwen models on the training splits of GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). For MBPP, we train a separate task-specific MLP monitor for each evaluated model, using rollouts from 374 problems for training and a disjoint set of 90 problems for validation; the 500-problem test set is used only for final evaluation. Following Sec. 3.3, we divide these rollouts into prefixes and estimate the continuation reliability Q for each prefix. Implementation details of prefix construction, continuation sampling, and risk labeling are provided in the Appendix.

Models. We use the Qwen3 series as the frozen base models that instantiate the agent, covering 4B, 8B, 14B, and 32B parameter scales. For the mathematical benchmarks, we train both dataset-specific reflection monitors and a pooled reflection monitor trained on the union of GSM8K and MATH training data, and evaluate these monitors separately. For MBPP, we evaluate Qwen3-4B and Qwen3-8B with their corresponding separately trained MBPP monitors.

Evaluation tasks. We use vanilla generation as the main baseline and compare it with Self-Agent under the same decoding configuration. We evaluate mathematical reasoning on the full GSM8K test set, MATH-500 (Lightman et al., 2023b), and AIME 2025 (Mathematical Association of America, 2024), denoted as AIME, and evaluate code generation on the 500-problem MBPP test set (Austin et al., 2021).

4.2 Main Results

We first evaluate Self-Agent on mathematical reasoning by comparing it with vanilla generation across model scales and benchmarks. For each base model, we train a pooled reflection monitor using rollouts from GSM8K and MATH, then place the same frozen model inside the Self-Agent system for evaluation. In all main-result experiments, each trajectory is allowed to trigger at most one intervention. Once a prefix triggers diagnosis, the system either accepts one local repair and resumes generation, or rejects the repair and continues the original trajectory without further intervention. The gains reported in this section are therefore obtained under a strictly bounded amount of additional computation.

Table 1: Main results across model scales and mathematical reasoning benchmarks.

		Qwen3-4B		Qwen3-8B		Qwen3-14B		Qwen3-32B	
Dataset	**Metric**	Vanilla	**Self-Agent**	Vanilla	**Self-Agent**	Vanilla	**Self-Agent**	Vanilla	**Self-Agent**
GSM8K	Acc.	86.1%	**89.5%**	93.4%	**94.1%**	95.4%	**95.5%**	95.5%	95.2%
	Δ	0.0%	**+3.5%**	0.0%	**+0.7%**	0.0%	**+0.1%**	0.0%	-0.2%
	Rescue (↑)	–	**32.1%**	–	**16.1%**	–	**6.6%**	–	**11.7%**
	Harm (↓)	–	**1.1%**	–	**0.4%**	–	**0.2%**	–	**0.8%**
MATH-500	Acc.	78.4%	**81.6%**	80.2%	**83.8%**	81.8%	**83.2%**	82.0%	**82.8%**
	Δ	0.0%	**+3.2%**	0.0%	**+3.6%**	0.0%	**+1.4%**	0.0%	**0.8%**
	Rescue (↑)	–	**16.7%**	–	**22.2%**	–	**9.9%**	–	**11.1%**
	Harm (↓)	–	**0.5%**	–	**1.0%**	–	**0.5%**	–	**1.5%**
AIME	Acc.	60.0%	**63.3%**	63.3%	**70.0%**	73.3%	**76.7%**	76.7%	**80.0%**
	Δ	0.0%	**+3.3%**	0.0%	**+6.7%**	0.0%	**+3.3%**	0.0%	**+3.3%**
	Rescue (↑)	–	**16.7%**	–	**27.3%**	–	**12.5%**	–	**28.6%**
	Harm (↓)	–	**5.6%**	–	**5.3%**	–	**0.0%**	–	**4.3%**

For each model and dataset, we report final-answer accuracy, the accuracy change over vanilla generation, the rate of rescued trajectories among baseline-wrong examples, and the rate of harmed trajectories among baseline-correct examples. Table 1 summarizes the results.

Comparison with Inference-Time Baselines. We further compare Self-Agent with two inference-time baselines in Table 2. Self-consistency samples multiple diverse reasoning trajectories and selects the most consistent final answer, typically through majority voting (Wang et al., 2023). Self-Refine improves an initial response through model-generated feedback and subsequent revision (Madaan et al., 2023). We use a single-pass instantiation of Self-Refine: the model critiques the complete vanilla solution without access to the reference answer and rewrites it only when an explicit global error is identified.

For a fair cost comparison with Self-Agent, we construct budget-matched variants of SC@3 and SC@5. Specifically, we randomly select a subset of examples for SC@3 or SC@5, while unselected examples retain their sample-1 responses. The selection process does not use the question content, the sample-1 response, or its correctness. We set the routing probability so that the expected token cost matches that of Self-Agent. All random assignments reuse the same set of pre-generated reasoning trajectories. We evaluate 20,000 independent random routing assignments; the point estimates in Table 2 are the resulting mean accuracies, while the brackets denote the 2.5th–97.5th percentiles. These intervals therefore capture only the variability introduced by random budget allocation, rather than model-generation variability. The corresponding full-budget results, where SC@3 or SC@5 is applied to every example, are reported in Appendix A.2.

Table 2: Comparison with inference-time baselines on Qwen3-4B and Qwen3-8B. Entries report final-answer accuracy (%); Self-Refine and Self-Agent additionally report token-cost multipliers relative to vanilla generation. Matched SC@3 and SC@5 use the same expected token budget as Self-Agent in the corresponding setting. Brackets give the 2.5th–97.5th percentiles over 20,000 random routing assignments. Bold denotes the best result, and underlining denotes the strongest competing baseline by accuracy.

Model	**Dataset**	**Vanilla Acc.**	**Matched SC@3 Acc. [95% interval]**	**Matched SC@5 Acc. [95% interval]**	**Self-Refine Acc. / Cost**	**Self-Agent Acc. / Cost**
Qwen3-4B	GSM8K	86.1	<u>87.5</u> [87.0, 88.0]	86.9 [86.4, 87.4]	85.9 / 3.13×	**89.5 / 1.55**×
	MATH-500	78.4	79.2 [78.6, 79.8]	79.0 [78.4, 79.6]	<u>79.4</u> / 4.25×	**81.6 / 1.72**×
	AIME	60.0	60.0 [56.7, 63.3]	<u>60.6</u> [60.0, 63.3]	56.7 / 2.31×	**63.3 / 1.76**×
Qwen3-8B	GSM8K	93.4	<u>93.8</u> [93.5, 94.2]	93.7 [93.4, 94.0]	90.5 / 4.71×	**94.1 / 1.70**×
	MATH-500	80.2	81.0 [80.0, 81.8]	81.0 [80.4, 81.8]	<u>81.2</u> / 3.54×	**83.8 / 1.70**×
	AIME	63.3	<u>65.6</u> [63.3, 66.7]	<u>65.6</u> [63.3, 70.0]	60.0 / 2.14×	**70.0 / 2.31**×

Code Generation on MBPP. To examine whether selective online repair extends beyond mathematical reasoning, we additionally evaluate Self-Agent on the 500-problem MBPP test set (Austin et al., 2021). For each model, we use an MBPP-specific MLP monitor trained on 374 problems and validated on a disjoint set of 90 problems, with no overlap with the test set. As shown in Table 3, Self-Agent improves execution-based Pass@1 from 63.2% to 68.0% for Qwen3-4B and from 66.0% to 68.6% for Qwen3-8B. These improvements correspond to net gains of 24 and 13 solved problems, respectively, showing that the rescue–harm trade-off remains favorable on code generation tasks.

Table 3: Results on the 500-problem MBPP test set. Pass@1 is measured by execution-based correctness, and Δ denotes the absolute improvement in percentage points. Rescue and harm rates use baseline-wrong and baseline-correct examples as denominators, respectively.

Model	Vanilla Pass@1	Self-Agent Pass@1	Δ Pass@1	Rescued (Rate)	Harmed (Rate)	Net Gain
Qwen3-4B	63.2% (316/500)	**68.0% (340/500)**	**+4.8**	28 (15.2%)	4 (1.3%)	**+24**
Qwen3-8B	66.0% (330/500)	**68.6% (343/500)**	**+2.6**	21 (12.4%)	8 (2.4%)	**+13**

4.3 Additional Analysis

We ablate each module of Self-Agent on Qwen3-4B GSM8K to verify that all three components contribute to the overall result. In these experiments, k denotes the percentage of riskiest prefixes selected for intervention by the monitor.

Hidden States vs. Uncertainty Signals. We compare the hidden-state MLP monitor with monitors trained on 13 token-level uncertainty features (entropy, top-1 probability, logit gap, etc.). Table 4 shows that hidden states consistently outperform uncertainty-only signals. The gap is especially large for Qwen3-8B, where the hidden-state monitor reaches 0.832 AUROC while the best uncertainty model reaches only 0.684. This confirms that prefix risk is not captured by local token hesitation alone; hidden states encode a richer summary of whether the current reasoning state is likely to derail, beyond what can be recovered from token-level uncertainty statistics alone (Setlur et al., 2024). We also ablate the depth and width of the monitor MLP in Appendix B.3, which shows that the chosen single-hidden-layer design is already sufficient.

Table 4: Monitor ablation comparing hidden-state risk prediction with uncertainty-based signals on pooled GSM8K+MATH data.

Model	Monitor	AUROC	AUPRC	GSM8K AUROC	MATH AUROC
Qwen3-4B	Hidden State MLP	**0.793** $\pm$0.010	**0.706** $\pm$0.020	**0.727**	**0.779**
	Uncertainty MLP	0.733 $\pm$0.010	0.654 $\pm$0.012	0.626	0.727
	Uncertainty LR	0.730 $\pm$0.010	0.656 $\pm$0.015	0.614	0.729
Qwen3-8B	Hidden State MLP	**0.832** $\pm$0.010	**0.841** $\pm$0.012	**0.867**	**0.782**
	Uncertainty MLP	0.684 $\pm$0.003	0.672 $\pm$0.014	0.636	0.726
	Uncertainty LR	0.676 $\pm$0.003	0.662 $\pm$0.012	0.628	0.722

Cost Analysis. We examine the extra computation introduced by Self-Agent. In all experiments, each trajectory is allowed to trigger at most one intervention, so the online overhead is strictly bounded by at most one diagnosis and one repaired continuation. Since the reflection monitor itself is lightweight, the dominant cost comes from the diagnostic call and, when accepted, the resumed continuation from the repaired state.

Figure 3 highlights three different cost-performance regimes. On GSM8K, the smaller 4B model benefits the most from added intervention budget, while the 8B model shows diminishing returns: more triggered diagnosis increases token cost substantially, but only recovers a few additional failures. On MATH-500, both models lie on a more favorable frontier, and the 8B model is consistently more

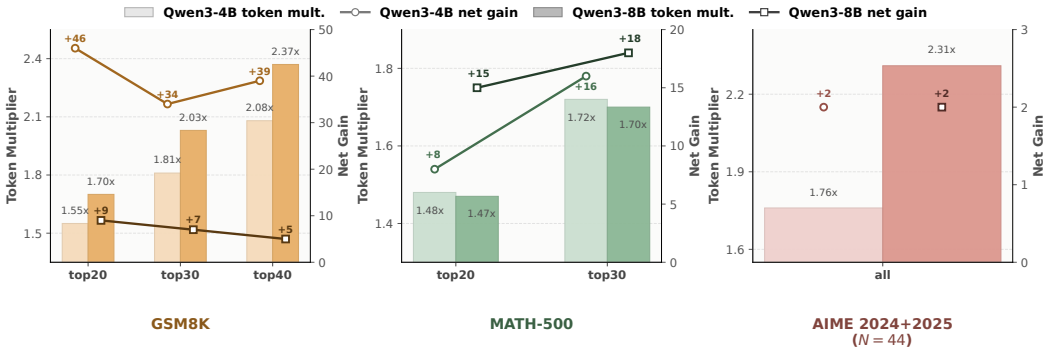


Figure 3: Cost-performance trade-offs of Self-Agent across datasets. Bars show token multiplier, and lines show net gain, with Qwen3-4B and Qwen3-8B plotted together under the same trigger budget. The AIME panel pools the 30 AIME 2025 problems with 14 AIME 2024 problems (N=44) to obtain a more stable cost estimate; the main accuracy results use AIME 2025 only.

efficient than 4B at similar token multipliers. The pooled AIME 2024–2025 cost evaluation behaves differently from both. Because every example is diagnosed, the cost is driven by long trajectories rather than by repeated triggering, and both models gain only modestly despite noticeably higher overhead. Detailed operating-point tables and token breakdowns are provided in Appendix C. Overall, the figure shows that Self-Agent improves reasoning under a tightly bounded intervention budget, but the most cost-effective operating point still depends strongly on the benchmark and the base-model scale.

Qualitative Analysis. Three representative Qwen3-8B GSM8K cases show that Self-Agent can repair local arithmetic mistakes but may over-correct under ambiguous interpretations. Full trajectories are provided in Appendix D.

5 Conclusion

We introduce Self-Agent, a lightweight framework for online reliability in a single frozen language-model agent. It trains a small reflection monitor from ordinary rollouts, invokes the frozen model in a separate diagnostic context for risky prefixes, and commits only bounded local repairs. Across Qwen3 models from 4B to 32B, Self-Agent improves mathematical reasoning on GSM8K, MATH-500, and AIME under a single-intervention budget; on MBPP, it also improves execution-based Pass@1 for the 4B and 8B models. These results show that identifying and repairing risky intermediate states can improve reasoning without retraining the generator or relying on a larger multi-agent scaffold. Future work should study broader domains, stronger cross-model transfer, and more robust online monitoring and repair.

Limitations

Our evaluation focuses on mathematical reasoning and one code-generation benchmark, all with automatically verifiable outcomes, so it does not establish the effectiveness of Self-Agent on open-ended tasks where correctness is ambiguous or difficult to verify. The reflection monitor is trained from outcome-level rollout labels, and the current system permits only one intervention per trajectory; its risk estimates and repair policy may therefore transfer poorly to other domains or multi-intervention settings. Extending Self-Agent to broader tasks, stronger cross-model transfer, and production-scale deployment remains important future work.

Ethics Statement

This work studies reliability interventions for language-model reasoning using publicly available benchmarks and model-generated trajectories. We did not conduct human-subject research or collect personal or sensitive information. Although Self-Agent can improve the accuracy of model outputs, it does not guarantee correctness: both the diagnostic module and the commit gate can introduce errors, and the appearance of self-correction may encourage unwarranted trust in an incorrect answer. We therefore report both rescued and harmed cases, quantify the additional inference cost, and limit our empirical claims to the mathematical-reasoning and code-generation settings evaluated in this paper. Deployment in high-stakes or open-ended domains would require domain-specific validation, human oversight, and safeguards beyond those studied here. Because a domain-general repair mechanism could also strengthen undesirable model behavior, it should be used in accordance with applicable laws, data and model licenses, and domain-specific safety requirements. We report our methods and results as transparently as possible and take responsibility for the accuracy and integrity of this work.

AI Use Statement

Generative AI tools were used for two purposes in this work. First, they assisted with language polishing and manuscript formatting. Second, the evaluated Qwen models generated the reasoning rollouts and sampled continuations used to construct synthetic prefix-level supervision for training the reflection monitors, following the procedures described in the paper. The model-generated training data were processed using the corresponding task-level outcome verifiers. The authors reviewed all AI-assisted text and verified the experimental procedures, results, citations, and claims. The authors take full responsibility for the final content of this work, including all text, claims, data, and artifacts.

Reproducibility Statement

The Self-Agent implementation is specified in Section 3, including the reflection monitor, separate-context diagnosis, and commit gate. Dataset construction and splits are described in Section 4.1 and Appendix A, which also reports the decoding and evaluation hyperparameters. Appendix B.3 details prefix construction, continuation sampling, risk labeling, monitor training, and architecture choices; Appendices B.1 and C document threshold selection and token-cost accounting; and Appendix D provides the diagnostic prompts and representative trajectories. This work makes no formal theorem or proof claims, so no separate proof appendix is required.

REFERENCES

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.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI Conference on Artificial Intelligence*, 2024.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: process supervision without process. *Advances in Neural Information Processing Systems*, 37:27689–27724, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, 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 *International Conference on Learning Representations*, 2024.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. V-star: Training verifiers for self-taught reasoners. In *Conference on Language Modeling*, 2024.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.

Hyunseok Lee, Seunghyuk Oh, Jaehyung Kim, Jinwoo Shin, and Jihoon Tack. Revise: Learning to refine at test-time via intrinsic self-verification. *arXiv preprint arXiv:2502.14565*, 2025.

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

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023a.

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 *The twelfth international conference on learning representations*, 2023b.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Mathematical Association of America. 2024–25 AIME thresholds are available, 2024. URL https://maa.org/news/aime-thresholds-are-available/. Accessed: 2026-05-06.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. In *International Conference on Learning Representations*, 2025.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. In *Advances in Neural Information Processing Systems*, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

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

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*, 2024.

John Yang, Kilian Leret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents, 2025a.

John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*, 2025b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b.

Fei Yu, Anningzhe Gao, and Benyou Wang. Outcome-supervised verifiers for planning in thematical reasoning. *CoRR, abs/2311.09724*, 2023.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. In *International Conference on Machine Learning*, 2024.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. In *International Conference on Machine Learning*, 2024.

A Experimental Setups

A.1 Decoding and Evaluation Settings

This appendix summarizes the decoding and evaluation setups used throughout the paper.

GSM8K. We use the thinking mode and set temperature to 0.65 with a maximum length of 16,384 tokens for ordinary trajectory generation. For separate-context diagnosis, we use temperature 0 and a maximum length of 8,192 tokens.

MATH-500. We disable the thinking mode because Qwen models already obtain very high baseline accuracy under thinking-mode generation, which leaves limited room for measuring repair effects. We use the same decoding settings as GSM8K for ordinary generation and diagnosis.

AIME. We use the thinking mode and set temperature to 0.6 with maximum length of 32,000 tokens for both ordinary generation and separate-context diagnosis. Main accuracy results use the 30 AIME

2025 problems. To reduce variance in the cost estimate, the cost analysis additionally includes 14 AIME 2024 problems, yielding a 44-problem AIME 2024–2025 pool.

MBPP. We train a separate task-specific MLP monitor for each evaluated model. We use 374 problems to train the monitor and a disjoint set of 90 problems for validation, while reserving the standard 500-problem test set exclusively for final evaluation. We report execution-based Pass@1 on the test set. We disable the thinking mode and set temperature to 0.65, top-p to 0.9, and the maximum generation length to 1,024 tokens.

A.2 Full-Budget Self-Consistency

Table 5 complements the budget-matched comparison in Table 2 by applying SC@3 or SC@5 to every evaluation example. No random routing or Monte Carlo averaging is used in this setting: each prediction is obtained by majority vote over all three or five pre-generated trajectories. Token costs are normalized by the corresponding sample-1 Vanilla run.

Table 5: Full-budget self-consistency results on GSM8K (N=1,319), MATH-500 (N=500), and AIME 2025 (N=30). Unlike the budget-matched variants in Table 2, SC@3 and SC@5 are applied to every example. For Vanilla and self-consistency, the number of correct answers is shown in parentheses. Cost is the token multiplier relative to Vanilla, and Net is the increase in correct answers over Vanilla. Bold denotes the highest accuracy in each row.

		Vanilla	Self-Agent	Full-budget SC@3		Full-budget SC@5	
Model	**Dataset**	**Acc. (Correct)**	**Acc. / Cost**	**Acc. (Correct) / Cost**	**Net**	**Acc. (Correct) / Cost**	**Net**
Qwen3-4B	GSM8K	86.1 (1135)	89.5 / 1.55×	91.3 (1204) / 2.98×	+69	**92.3** (1217) / 5.07×	+82
	MATH-500	78.4 (392)	**81.6** / 1.72×	80.6 (403) / 3.01×	+11	**81.6** (408) / 5.03×	+16
	AIME	60.0 (18)	**63.3** / 1.76×	60.0 (18) / 3.00×	0	**63.3** (19) / 5.01×	+1
Qwen3-8B	GSM8K	93.4 (1232)	94.1 / 1.70×	94.5 (1247) / 3.00×	+15	**95.0** (1253) / 4.99×	+21
	MATH-500	80.2 (401)	83.8 / 1.70×	82.4 (412) / 3.03×	+11	**84.8** (424) / 5.04×	+23
	AIME	63.3 (19)	**70.0** / 2.31×	66.7 (20) / 2.96×	+1	**70.0** (21) / 4.80×	+2

Full-budget self-consistency obtains its largest gains on GSM8K, especially for Qwen3-4B, but requires approximately $3\times$ or $5\times$ the Vanilla token cost, compared with $1.55\times$ for Self-Agent. On MATH-500, SC@5 ties Self-Agent for Qwen3-4B and exceeds it by 1.0 percentage point for Qwen3-8B, while using roughly three times the token cost of Self-Agent. On AIME, SC@5 ties Self-Agent for both model sizes despite its higher cost. Thus, uniformly allocating additional samples can improve raw accuracy, whereas the budget-matched results in Table 2 isolate performance under comparable expected compute.

B Additional Results and Ablations

This section provides several further analyses. Appendix B.1 studies how the monitor threshold affects the rescue-harm trade-off across trigger budgets. Appendix B.3 reports monitor architecture ablations, dataset-specific versus pooled training, per-feature uncertainty breakdowns, separate-context versus in-context diagnosis, and difficulty-stratified ablations of both the diagnostic module and the commit gate. Appendix B.2 tests cross-model generalization by transferring an 8B monitor to a 14B base model without retraining.

Performance Across Difficulty Analysis. Table 6 further shows that the benefit of Self-Agent depends on where the remaining errors are recoverable. For Qwen3-4B on GSM8K, the gain increases from easy to hard splits, rising from +1.7% to +5.9%, which suggests that local repair is especially useful when the base model still has many fixable reasoning mistakes. On MATH-500, however, the hard split contains many baseline errors but yields only a small net gain, while the mixed split shows a large percentage gain because it has few examples and several rescued cases. Qwen3-8B shows a similar distinction: it gains little on the hard GSM8K and MATH-500 splits, even though those splits contain many errors, indicating that some failures are beyond local repair rather than merely unmonitored mistakes. Overall, the difficulty analysis suggests that Self-Agent is most

effective on mistakes that the frozen model can diagnose and repair once the risky prefix is isolated; as tasks become too hard, monitor noise and unrecoverable errors increasingly limit the net gain.

Table 6: Difficulty-stratified analysis for Qwen3-4B and Qwen3-8B across GSM8K, MATH-500, and AIME. *Errors* denotes the number of examples answered incorrectly by the baseline. Rescued and Harmed count examples where Self-Agent corrected or introduced errors, respectively. *Net* is Rescued − Harmed, and ↑ Acc. (%) is the absolute accuracy gain within the split.

Model	Metric	GSM8K			MATH-500			AIME
		easy	*mixed*	*hard*	*easy*	*mixed*	*hard*	*2025*
Qwen3-4B	#. Examples	655	**411**	253	**446**	14	40	**30**
	Errors	37	**51**	96	**63**	8	37	**12**
	Rescued	15	**24**	20	**13**	3	2	**2**
	Harmed	4	**4**	5	**1**	0	1	**1**
	Net	+11	**+20**	+15	**+12**	+3	+1	**+1**
	↑ Acc. (%)	1.7	**4.9**	5.9	**2.7**	21.4	2.5	**3.3**
Qwen3-8B	#. Examples	**655**	411	253	**446**	14	40	**30**
	Errors	**10**	18	59	**53**	8	38	**11**
	Rescued	**5**	5	4	**21**	1	0	**3**
	Harmed	**0**	2	3	**4**	0	0	**1**
	Net	**+5**	+3	+1	**+17**	+1	0	**+2**
	↑ Acc. (%)	**0.8**	0.7	0.4	**3.8**	7.1	0.0	**6.7**

Gate Variants. We compare the default commit gate with two alternatives: *no gate* (accept every repair) and a *stricter gate* (accept only highly local arithmetic fixes). Table 7 shows that the default gate provides the best operating point: at k=20, it achieves 32.1% rescue with only 1.1% harm, versus 30.4% rescue / 1.7% harm without the gate. The stricter gate over-filters, reducing rescue to 21.7%. The default gate balances coverage and reliability across all trigger budgets.

Table 7: Commit-gate ablation on Qwen3-4B GSM8K (k: top-% riskiest prefixes). The default gate is compared with ungated repair and a stricter arithmetic-only gate.

Gate Variant	k %	Δ Acc.	Rescue %	Harm %	Net Gain
Default gate	20	**+3.5**	**32.1**	**1.1**	**+46**
	30	**+2.6**	**34.2**	**2.6**	**+34**
	40	**+3.0**	**39.7**	3.0	**+39**
Without gate	20	+2.8	30.4	1.7	+37
	30	+1.9	33.7	3.3	+25
	40	+2.1	39.1	3.9	+28
Stricter gate	20	+2.0	21.7	1.2	+26
	30	+1.2	25.5	2.7	+16
	40	+1.9	31.0	**2.8**	+25

B.1 Threshold Study

We study how the monitor threshold changes the rescue-harm trade-off. In our implementation, each threshold corresponds to a trigger budget, which controls how many prefixes are sent to the diagnostic module. A larger trigger budget gives Self-Agent more chances to rescue failed trajectories, but it also sends more uncertain cases to the gate and can therefore increase harm. Since the current AIME setting uses trigger-all diagnosis rather than a threshold sweep, this analysis focuses on GSM8K and MATH-500.

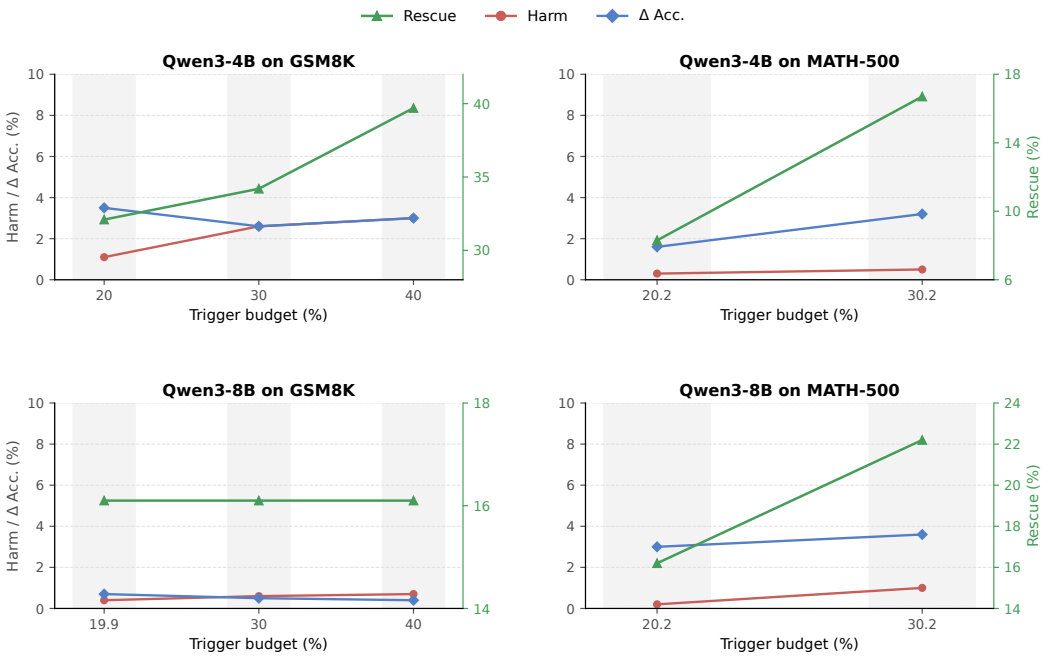


Figure 4: Threshold study for Self-Agent. The x-axis is the realized trigger budget. Rescue and harm are measured relative to baseline wrong and baseline correct trajectories, respectively, while net gain counts rescued cases minus harmed cases.

Figure 4 shows that increasing the trigger budget is not always better. On Qwen3-4B GSM8K, the rescue rate rises from 32.1% to 39.7% as k increases from 20 to 40, but the harm rate also increases from 1.1% to 3.0%. The best net improvement is therefore obtained at the smaller budget, where the monitor already covers many recoverable errors while avoiding many safe trajectories. Qwen3-8B GSM8K shows an even clearer pattern: the rescue rate stays flat at 16.1%, while harm grows as more prefixes are triggered, so a looser threshold only reduces the net gain.

MATH-500 behaves differently. For both Qwen3-4B and Qwen3-8B, moving from k=20 to k=30 increases rescue more than harm, so the net gain improves. This suggests that the best threshold depends on the base model and the benchmark. In practice, we select the operating point by the rescued-minus-harmed objective in Section 3.1, rather than by trigger rate alone.

Rescue-Harm Trade-off. We analyze the trade-off induced by the intervention threshold. For each operating point, we report how often Self-Agent triggers diagnosis, how often the gate accepts a repair, and how the accepted repairs decompose into rescued and harmed cases.

Table 9: Cross-model generalization from an 8B reflection monitor to a 14B base model on MATH-500. The 8B monitor is reused without retraining; 14B rollouts are generated by the frozen 14B model, while prefix hidden states are re-encoded by the 8B model so that the 4096-dimensional MLP can be applied directly.

Setting	Split	N	Errors	GatePass	Rescued	Harmed	Net / ΔAcc.
Top-20 + Gate V1	overall	500	91	23	7	2	+5 / +1.0%
	easy	446	54	—	**7**	**0**	**+7 / +1.6%**
	mixed	14	5	—	0	0	0 / 0.0%
	hard	40	32	—	0	2	-2 / -5.0%
Top-30 + Gate V1	overall	500	91	31	**9**	3	**+6 / +1.2%**
	easy	446	54	—	**9**	1	**+8 / +1.8%**
	mixed	14	5	—	0	0	0 / 0.0%
	hard	40	32	—	0	2	-2 / -5.0%

Table 8: Rescue-harm trade-off under different trigger budgets. Trigger rate is measured over all test examples, gate pass rate is measured among triggered examples, rescue rate is measured among baseline-wrong examples, and harm rate is measured among baseline-correct examples. Net gain is rescued minus harmed.

Model	Dataset	Budget	Trigger	Gate	Rescued	Harmed	Net	Rescue	Harm
Qwen3-4B	**GSM8K**	**top20**	**20.0%**	**93.2%**	**59**	**13**	**+46**	**32.1%**	**1.1%**
Qwen3-4B	GSM8K	top30	30.0%	89.4%	63	29	+34	34.2%	2.6%
Qwen3-4B	GSM8K	top40	40.0%	91.3%	73	34	+39	39.7%	3.0%
Qwen3-4B	MATH-500	top20	20.2%	55.4%	9	1	+8	8.3%	0.3%
Qwen3-4B	**MATH-500**	**top30**	**30.2%**	**55.6%**	**18**	**2**	**+16**	**16.7%**	**0.5%**
Qwen3-8B	**GSM8K**	**top20**	**19.9%**	**19.8%**	**14**	**5**	**+9**	**16.1%**	**0.4%**
Qwen3-8B	GSM8K	top30	30.0%	14.9%	14	7	+7	16.1%	0.6%
Qwen3-8B	GSM8K	top40	40.0%	12.3%	14	9	+5	16.1%	0.7%
Qwen3-8B	MATH-500	top20	20.2%	45.5%	16	1	+15	16.2%	0.2%
Qwen3-8B	**MATH-500**	**top30**	**30.2%**	**43.0%**	**22**	**4**	**+18**	**22.2%**	**1.0%**

B.2 Cross-Model Generalization

We test whether a reflection monitor trained on Qwen3-8B can be reused on a stronger 14B base model without retraining. Because the 8B monitor expects 4096-dimensional hidden states while Qwen3-14B produces 5120-dimensional states, we adopt a frozen transfer protocol: the reasoning rollout is still generated by Qwen3-14B, but each triggered prefix is re-encoded by Qwen3-8B so that the original 8B MLP can be applied directly. This isolates cross-model generalization at the level of prefix-risk detection rather than changing the monitor architecture itself.

The transferred 8B monitor remains weakly but consistently useful on MATH-500. At Top-20, it rescues 7 of the 91 baseline errors while harming only 2 correct trajectories, yielding a net gain of +5 and a +1.0 point accuracy improvement. At Top-30, the gain rises slightly to +6. These gains are smaller than in the matched 8B setting, where the same operating points recover 16 and 22 trajectories, respectively. This gap is expected: the 14B baseline is already stronger, leaving fewer recoverable failures, and the transferred threshold is calibrated on the 8B hidden-state distribution rather than the 14B reasoning dynamics. The difficulty breakdown makes the pattern clearer. All successful transfers occur on the easy split, while the hard split produces no rescued cases and a small number of harms. This suggests that the learned notion of prefix risk has some cross-model stability, but its practical value degrades when the downstream model becomes both stronger and distributionally different.

B.3 Monitor and Repair Ablations

Prefix-level supervision. The monitor is trained from ordinary model rollouts without step-level human annotation. For each rollout, we construct prefix candidates at several step boundaries,

Table 11: Depth and width ablation of the reflection monitor on Qwen3-4B MATH-500. A single-hidden-layer MLP is sufficient: linear probing underfits, deeper MLPs degrade, and increasing width beyond 128 brings little benefit.

Model	Layers	Params	Train AUROC	Val AUROC	Val AUPRC	Brier	Gap	P@20	R@20
Linear	0	2,561	0.9339	0.7463 ± 0.003	0.7398	0.2377	0.1876	0.766	0.634
MLP-1h64	1	163,969	0.9125	**0.8069 ± 0.012**	**0.8033**	**0.1813**	0.1056	**0.849**	0.608
MLP-1h128	1	327,937	**0.9248**	0.8049 ± 0.012	0.8016	0.1831	0.1200	**0.849**	**0.616**
MLP-1h256	1	655,873	0.9166	0.8059 ± 0.014	0.8034	0.1826	**0.1107**	0.848	0.618
MLP-2layer	2	1,377,025	0.9155	0.7979 ± 0.013	0.7949	0.1925	0.1176	0.833	0.602
MLP-3layer	3	1,475,585	0.9186	0.7964 ± 0.011	0.7917	0.1926	0.1223	0.819	0.592
MLP-10layer	10	1,836,001	0.6094	0.6018 ± 0.020	0.6001	0.2507	0.0076	0.607	0.669

including relative positions in the trajectory, early online steps, and one random valid step boundary. For each prefix, we sample continuations from the same frozen model and estimate whether the prefix is likely to recover to a correct final answer. Prefixes whose continuations rarely lead to a correct answer are labeled as high risk, while prefixes whose continuations are usually correct are labeled as low risk. Ambiguous prefixes are discarded. Table 10 summarizes the resulting supervision used for the pooled monitors.

Table 10: Prefix-level supervision for reflection monitor training. CHECK=1 denotes high-risk prefixes and CHECK=0 denotes low-risk prefixes.

Model	Dataset	Raw	Filtered	CHECK=1	CHECK=0	Dropped
Qwen3-4B	GSM8K	15285	14999	3657	11342	286
Qwen3-4B	MATH	15581	14558	7391	7167	1023
Qwen3-4B	Total	30866	29557	11048	18509	1309
Qwen3-8B	GSM8K	11498	11498	5083	6415	0
Qwen3-8B	MATH	16839	15633	7897	7736	1206
Qwen3-8B	Total	28337	27131	12980	14151	1206

The label distribution differs across datasets and model scales. GSM8K prefixes from Qwen3-4B are mostly low risk, while MATH prefixes are nearly balanced. For Qwen3-8B, GSM8K becomes harder to separate by final correctness alone because the base model is already strong on many examples, but the pooled data remains close to balanced after filtering. We therefore use class balancing and difficulty-aware sample weights when training the monitor.

Monitor architecture ablation. We also ablate the depth and width of the reflection monitor on Qwen3-4B MATH-500. Table 11 shows that introducing a single hidden layer gives a large gain over linear probing, but further increasing depth is harmful. The three one-hidden-layer variants with width 64, 128, and 256 perform almost identically, which indicates that the main benefit comes from a small amount of nonlinearity rather than from scale. Deeper networks degrade monotonically, and the 10-layer model nearly collapses, suggesting that small prefix-supervision datasets do not support large monitor capacity.

Dataset-specific versus pooled training. We compare monitors trained on one dataset with monitors trained on the union of GSM8K and MATH. Each monitor is a lightweight MLP over the last hidden state of a prefix, with a 2560-dimensional input for Qwen3-4B and a 4096-dimensional input for Qwen3-8B. We train with question-level splits and report three-seed cross-validation results in Table 12.

Table 12: Dataset-specific versus pooled monitor training. Each row compares a monitor trained on the matching dataset with a monitor trained on GSM8K+MATH, evaluated on the same validation subset.

Model	Training data	Overall AUROC	Overall AUPRC	GSM8K AUROC	MATH AUROC
Qwen3-4B	GSM8K only	–	–	**0.713** ±0.018	–
Qwen3-4B	MATH only	–	–	–	**0.808** ±0.011
Qwen3-4B	GSM8K+MATH	**0.793** ±0.010	**0.706** ±0.020	0.727 ±0.025	0.779 ±0.026
Qwen3-8B	GSM8K only	–	–	0.856 ±0.001	–
Qwen3-8B	MATH only	–	–	–	0.773 ±0.021
Qwen3-8B	GSM8K+MATH	**0.832** ±0.010	**0.841** ±0.012	**0.867** ±0.004	**0.782** ±0.024

Pooled training gives a single monitor that remains competitive across both datasets. The pooled monitors obtain overall AUROC/AUPRC of 0.793/0.706 for Qwen3-4B and 0.832/0.841 for Qwen3-8B on the pooled validation splits. For Qwen3-4B, pooling improves GSM8K AUROC over the GSM8K-only monitor but slightly reduces MATH AUROC compared with the MATH-only monitor. For Qwen3-8B, pooling improves both dataset-specific AUROCs. This suggests that risk information is partly shared across mathematical reasoning datasets, although the benefit depends on the base model and label distribution.

Uncertainty feature breakdown. Within the uncertainty family, no single token-level statistic closes the gap with hidden states. Table 13 shows per-feature results. For Qwen3-4B, the strongest single-feature baselines are mean log probability and top-1 probability, but both remain well below the hidden-state monitor. For Qwen3-8B, the best single-feature AUROC is only 0.638, and the strongest uncertainty model remains far below the hidden-state monitor.

Table 13: Uncertainty-only ablation for the reflection monitor on pooled GSM8K+MATH data. We compare multi-feature uncertainty baselines with single-feature logistic-regression baselines.

Model	Monitor	AUROC	AUPRC	GSM8K AUROC	MATH AUROC
Qwen3-4B	**Uncertainty MLP**	**0.733** ±0.010	0.654 ±0.012	0.626	0.727
Qwen3-4B	**Uncertainty LR**	0.730 ±0.010	**0.656** ±0.015	0.614	**0.729**
Qwen3-4B	Mean Logprob only (LR)	0.696 ±0.020	0.607 ±0.018	0.623	0.703
Qwen3-4B	Top1 Prob only (LR)	0.697 ±0.011	0.589 ±0.018	0.568	0.675
Qwen3-4B	Entropy only (LR)	0.688 ±0.022	0.600 ±0.018	**0.627**	0.695
Qwen3-4B	Logit Gap only (LR)	0.652 ±0.011	0.495 ±0.013	0.550	0.609
Qwen3-8B	**Uncertainty MLP**	**0.684** ±0.003	**0.672** ±0.014	**0.636**	**0.726**
Qwen3-8B	**Uncertainty LR**	0.676 ±0.003	0.662 ±0.012	0.628	0.722
Qwen3-8B	Top1 Prob only (LR)	0.638 ±0.009	0.621 ±0.019	0.607	0.708
Qwen3-8B	Logit Gap only (LR)	0.625 ±0.010	0.581 ±0.014	0.624	0.642
Qwen3-8B	Mean Logprob only (LR)	0.625 ±0.011	0.634 ±0.010	0.529	0.705
Qwen3-8B	Entropy only (LR)	0.619 ±0.012	0.628 ±0.013	0.528	0.696

Separate vs. in-context diagnosis. We compare Self-Agent with a direct self-checking baseline that appends a verification prompt to the original trajectory instead of diagnosing in a separate context. Both methods use the same monitor and trigger on the same prefixes. Table 14 shows that direct self-checking causes substantially more harm: at k=20, Self-Agent improves accuracy by 3.5 points while self-checking gains only 1.1 points due to 32 harmed trajectories. Self-checking turns negative at k=30 and k=40, while Self-Agent remains positive. This supports the design choice of moving diagnosis out of the original trajectory, consistent with prior evidence that language models often fail to reliably self-correct while staying conditioned on their original flawed reasoning (Huang et al., 2023).

Table 14: Separate-context diagnosis vs. in-context self-checking on Qwen3-4B GSM8K (k: top-% riskiest prefixes selected for intervention). Both methods use the same reflection monitor and trigger on the same prefixes.

Method	k %	Δ Acc.	Rescue %	Harm %	Net Gain
Self-Agent (ours)	20	**+3.5**	**32.1**	**1.1**	**+46**
	30	**+2.6**	**34.2**	**2.6**	**+34**
	40	**+3.0**	**39.7**	**3.0**	**+39**
Self-Check	20	+1.1	25.5	2.8	+15
	30	−1.4	27.7	6.2	−19
	40	−2.6	32.1	8.2	−34

Diagnostic ablation by difficulty. Table 15 breaks the diagnostic comparison by difficulty. On hard examples, self-checking harms more correct trajectories than it rescues, giving a negative net gain, while Self-Agent still recovers 20 baseline errors with only 5 harmed cases.

Table 15: Difficulty-stratified diagnostic ablation at the Top-20 operating point on Qwen3-4B GSM8K.

Method	Split	N	Errors	Repaired	Rescued	Harmed	Net	Δ Acc.
Ours	**easy**	**655**	**37**	**72**	**15**	**4**	**+11**	**+1.7**
Self-check	easy	655	37	72	15	6	+9	+1.4
Ours	**mixed**	**411**	**51**	**85**	**24**	**4**	**+20**	**+4.9**
Self-check	mixed	411	51	85	19	10	+9	+2.2
Ours	**hard**	**253**	**96**	**107**	**20**	**5**	**+15**	**+5.9**
Self-check	hard	253	96	107	13	16	-3	-1.2

Gate ablation by difficulty. Table 16 shows the gate comparison by difficulty. On easy examples, ungated repair works well, but on mixed and hard examples the default gate provides the strongest result by filtering out lower-quality corrections.

Table 16: Difficulty-stratified gate ablation at the Top-20 operating point on Qwen3-4B GSM8K.

Variant	Split	N	Errors	Rescued	Harmed	Net
gate	easy	655	37	15	4	+11
gate	mixed	411	51	**24**	**4**	**+20**
gate	hard	253	96	**20**	**5**	**+15**
without gate	easy	655	37	**16**	3	**+13**
without gate	mixed	411	51	24	5	+19
without gate	hard	253	96	16	11	+5
stricter gate	easy	655	37	13	**2**	+11
stricter gate	mixed	411	51	18	4	+14
stricter gate	hard	253	96	9	8	+1

C COST DETAILS

All operating points. Table 17 reports the full cost table across all trigger budgets. All AIME cost statistics in this section use the pooled AIME 2024–2025 set (N=44), rather than the AIME 2025-only set used for the main accuracy comparison.

Table 17: Detailed cost table across all trigger budgets. Trigger rate is measured over all examples, accept rate is measured among triggered examples, and token multiplier is total tokens divided by baseline tokens. AIME cost statistics pool the 30 AIME 2025 problems with 14 AIME 2024 problems (N=44); the main accuracy results use AIME 2025 only.

Model	Dataset	Budget	Trigger	Accept	Net	Total Tokens	Token Mult.
Qwen3-4B	GSM8K	top20	20.0%	93.2%	+46	787,666	1.55x
	GSM8K	top30	30.0%	89.4%	+34	920,149	1.81x
	GSM8K	top40	40.0%	91.3%	+39	1,061,498	2.08x
	MATH-500	top20	20.2%	55.4%	+8	627,987	1.48x
	MATH-500	top30	30.2%	55.6%	+16	729,589	1.72x
	AIME 24–25	all	100.0%	70.5%	+2	1,327,455	1.76x
Qwen3-8B	GSM8K	top20	19.9%	19.8%	+9	707,777	1.70x
	GSM8K	top30	29.9%	14.9%	+7	847,042	2.03x
	GSM8K	top40	40.0%	12.3%	+5	985,942	2.37x
	MATH-500	top20	20.2%	45.5%	+15	711,044	1.47x
	MATH-500	top30	30.2%	43.0%	+18	823,599	1.70x
	AIME 24–25	all	100.0%	36.4%	+2	678,308	2.31x

Per-example token breakdown. Table 18 decomposes per-example tokens into baseline generation, diagnostic overhead, and repaired continuation.

Table 18: Per-example token breakdown of Self-Agent. Baseline is the ordinary rollout length, Diagnostic is the additional cost of the separate diagnostic call, Repair is the additional cost of continuing from the repaired state when a repair is accepted, and Overhead is Diagnostic plus Repair. AIME 24–25 pools AIME 2024 and 2025 (N=44).

Model	Dataset	Budget	Baseline	Diagnostic	Repair	Overhead	Total
Qwen3-4B	GSM8K	top20	386	128	83	211	597
	GSM8K	top30	386	192	119	311	698
	GSM8K	top40	386	257	162	419	805
	MATH-500	top20	847	303	106	409	1,256
	MATH-500	top30	847	453	160	613	1,459
	AIME 24–25	all	17,152	4,756	8,262	13,017	30,169
Qwen3-8B	GSM8K	top20	316	206	14	221	537
	GSM8K	top30	316	310	16	327	642
	GSM8K	top40	316	414	18	432	747
	MATH-500	top20	967	324	132	456	1,422
	MATH-500	top30	967	484	197	681	1,647
	AIME 24–25	all	6,677	6,059	2,680	8,739	15,416

Per-call token breakdown. Table 19 further separates diagnostic and repaired-continuation calls into input and output tokens.

Table 19: Per-call token breakdown for diagnosis (B) and repaired continuation (A). AIME 24–25 pools AIME 2024 and 2025 (N=44).

	Diagnosis (B)			Continuation (A)		
Config	**Input**	**Output**	**Total**	**Input**	**Output**	**Total**
4B GSM8K	521	120	641	280	163	443
4B MATH	900	600	1,500	506	444	951
4B AIME 24–25	4,156	600	4,756	8,684	3,042	11,726
8B GSM8K	436	600	1,036	245	121	366
8B MATH	1,002	600	1,602	566	516	1,083
8B AIME 24–25	5,459	600	6,059	3,393	3,977	7,370

D Prompts and Examples

D.1 Diagnostic Prompts

We first show the two diagnostic prompts used in our experiments. Prompt 1 is the default diagnosis prompt used throughout the main experiments. Prompt 2 is the stricter verifier used only in the gate ablation, where we intentionally tighten the acceptance criterion for local repair.

Prompt 1: Default Diagnosis Prompt

Role. You are a cautious reasoning diagnostician.

Inputs.

- `[Problem] {question}`
- `[Verified Reliable History] {safe_history}`
- `[Recently Suspicious Reasoning] {suspect_window}`

Task. Determine whether the recent reasoning contains a clear local error. If and only if a clear local error exists, provide one minimal corrected fact.

Rules.

- Do not solve the whole problem.
- Do not give the final answer.
- Only focus on the most recent local mistake.
- Be conservative: if unsure, set `DECISION=UNCERTAIN`.
- The corrected fact must be short and local.
- Do not rewrite more than the last 1–2 steps.

Output format. Output your diagnosis inside `<DIAGNOSIS>` tags in exactly this format:

```
<DIAGNOSIS> DECISION=NO_ERROR or UNCERTAIN or CONFIRMED_ERROR
CONFIDENCE=0.00 to 1.00 ERROR_LOCATION=last_step or last_2_steps
or none ERROR_TYPE=arithmetic or algebra or constraint or
variable_tracking or missing_case or none EVIDENCE=one short sentence
or none CORRECTED_FACT=one short sentence or none REWRITE_SCOPE=none or
last_step or last_2_steps </DIAGNOSIS>
```

Prompt 2: Strict Verifier for Gate Ablation

Role. You are a strict math error verifier. Your job is to check whether the shown reasoning contains a `CONCRETE, VERIFIABLE` mathematical mistake.

Inputs.

- `[Problem] {question}`
- `[Verified Reliable History] {safe_history}`
- `[Recently Suspicious Reasoning] {suspect_window}`

Task. Check `ONLY` for concrete mathematical errors in the suspicious reasoning above:

- Wrong arithmetic (e.g., `3+5=9`, `12*4=52`)
- Wrong algebraic manipulation (e.g., dividing both sides but flipping sign incorrectly)
- Using a wrong number from the problem statement
- Contradicting a fact established in the verified history

What does `NOT` count as an error.

- Reasoning that is incomplete or could be continued differently
- Steps that are correct but use a different valid approach
- Missing explanations or justifications (as long as the math is right)
- The solution not being finished yet

Verification constraint. You `MUST` verify each calculation yourself before claiming it is wrong. If you cannot find a specific wrong calculation, set `DECISION=NO_ERROR`.

Important. Most reasoning steps are actually correct. Default to `NO_ERROR` unless you find an undeniable mistake. Use `CONFIDENCE` below `0.90` if you have any doubt.

Output format. Output your diagnosis inside `<DIAGNOSIS>` tags in exactly this format:

```
<DIAGNOSIS> DECISION=NO_ERROR or UNCERTAIN or CONFIRMED_ERROR
CONFIDENCE=0.00 to 1.00 ERROR_LOCATION=last_step or last_2_steps
or none ERROR_TYPE=arithmetic or algebra or constraint or
variable_tracking or missing_case or none EVIDENCE=one short sentence
showing the specific wrong calculation, or none CORRECTED_FACT=one
short sentence with the correct calculation, or none REWRITE_SCOPE=none
or last_step or last_2_steps </DIAGNOSIS>
```

We then provide three representative Qwen3-8B GSM8K cases to illustrate both successful repair and systematic harm. In each card below, we retain the local prefix that triggered intervention, the structured diagnostic conclusion, and the resulting continuation after repair injection.

Rescued Case

Problem. Freda's 16-ounce canned tomatoes lose half their volume when cooked into sauce. A 32-ounce batch is made, and each can contains 3 tomatoes.

Baseline trajectory. The baseline incorrectly states that one 16-ounce can still produces 16 ounces of sauce, which leads to 2 cans and a final answer of 6.

Diagnostic output. `DECISION=CONFIRMED_ERROR, CONFIDENCE=1.00, ERROR_TYPE=arithmetic`. The diagnostic module identifies the local fact that one 16-ounce can should produce 8 ounces of sauce after losing half its volume.

Continued trajectory. After injecting the corrected fact, the model recomputes the number of cans as $32/8 = 4$, then reaches the correct final answer 12.

Outcome. This case shows the desired local-repair behavior: the intervention changes one mistaken fact, while leaving the rest of the solution structure intact.

Harmed Case 1

Problem. James hires a carriage from 5 PM to 9 PM, gets 1 hour free, pays $15 for the first paid hour, and "each hour after that is twice the cost."

Baseline trajectory. The baseline interprets the two later paid hours as both costing $30 and reaches the benchmark answer 75.

Diagnostic output. `DECISION=CONFIRMED_ERROR, CONFIDENCE=1.00`. The diagnostic module reads the statement as geometric doubling, so it rewrites the local fact from the third paid hour is also $30 to the third paid hour should be $60.

Continued trajectory. The repaired continuation recomputes the total as $15 + 30 + 60 = 105$, which is internally consistent under that reading but disagrees with the benchmark label.

Outcome. The harm does not come from random corruption. It comes from a stable but stricter semantic interpretation than the one implied by the benchmark answer.

Harmed Case 2

Problem. Tasha earns $80 from lemonade and mowing lawns. The second week, Alba's lawn is mowed five times as often as Joe's, and Joe pays $6 per mowing.

Baseline trajectory. The baseline reuses the same variable across weeks, derives total mowing income $6(x + 3x + 5x) = 54x$, and reaches the benchmark answer 26.

Diagnostic output. `DECISION=CONFIRMED_ERROR, CONFIDENCE=1.00, ERROR_TYPE=variable_tracking`. The diagnostic module argues that the second week's Joe count should be a separate variable rather than reusing the first week's x.

Continued trajectory. Once that correction is accepted, the continuation abandons the original compact structure and shifts into a different search process, eventually producing 8 instead of 26.

Outcome. This case shows another semantic mismatch: the repair is logically defensible in isolation, but it overrides the simplified interpretation expected by the benchmark.