Efficient Forecasting of Task Failures in LLM Agents through Adaptive Fault Injection

Vartika Sengar [*1] **Parth Thakkar** [*1] **Pranoy Panda** [*1] **Shrey Satapara** [1] **Emmy Liu** [2] **Vijay Viswanathan** [2] **Sho Takemori** [1] **Graham Neubig** [2] **Chaitanya Devaguptapu** [1]

Abstract

LLM agents perform long-horizon tasks where small errors can compound across tool calls and reasoning steps. We introduce **Task Failure Forecasting**, which predicts from a partial execution trace whether the current agent policy is likely to fail. Unlike retrospective failure analysis, this enables runtime intervention while recovery is still possible. The key challenge is separating recoverable mistakes from fatal ones, a distinction humans make with only 54% accuracy. To address this, we use **adaptive fault injection**: targeted perturbations of successful traces, guided by bandit-based prioritization and verified through rollouts. Trained on this synthetic supervision, a lightweight Qwen-3-8B forecaster achieves 78.01% weighted F1, outperforms Claude-4.5 Sonnet by 18 points at 100× lower cost, and improves agent success by 2–16 % points across diverse benchmarks.

1. Introduction

Recent LLM agents operate over long horizons (Mialon et al., 2023; Jimenez et al., 2023), use external tools (Yao et al., 2022; Fourney et al., 2024), and interact with evolving environments. As tasks unfold through many sequential decisions, not all mistakes have the same consequence: some can be corrected by later actions, while others make successful completion unlikely under the current policy. Distinguishing these cases during execution is important because it determines whether the agent should continue normally or allocate intervention compute, such as reflection (Shinn et al., 2023), resampling (Wang et al., 2022), or human handoff. However, most agent failure analysis remains *post-hoc*. Human annotators inspect failed traces and identify root causes retrospectively (Zhang et al., 2025c), while LLM-as-judge systems and recent fault-localization methods (Zhang et al., 2025a; Kong et al., 2025) also operate after execution has finished. These methods support attribution and debugging, but do not answer the inference-time question: *given the current partial trace, is continued execution under the present policy likely to succeed?* This question is difficult even for humans: expert annotators achieve only 54% accuracy when distinguishing recoverable from high-risk errors on partial traces.

We study this problem as ***Task Failure Forecasting***: predicting, from a partial trace $\tau_{\leq t}$, whether continued execution under policy π is likely to fail. Equivalently, we estimate $f_\theta(\tau_{\leq t}) := P(Y(\tau) = 1 \mid \tau_{\leq t}, \pi)$, where $Y(\tau)$ denotes terminal outcome and $Y(\tau) = 1$ indicates failure. Failure risk is policy-conditioned rather than absolute: the same prefix may be recoverable for a stronger policy but high-risk for a weaker one. Task Failure Forecasting differs from fault localization and process supervision (Zhang et al., 2025d) in both timing and target. Fault localization identifies the cause of an already completed failure, while process supervision judges local step quality. In contrast, our goal is to predict before termination whether the current trace remains recoverable. This distinction matters because local correctness is not equivalent to future reachability: an incorrect step may be recoverable, while a plausible step may place the agent on a path from which the current policy rarely recovers.

Training such a forecaster is challenging. Monte Carlo rollout (Luo et al., 2024; Wang et al., 2024) can estimate success probability from each prefix, but is computationally expensive for long agent traces (Jimenez et al., 2023; Zhang et al., 2025c). Learning directly from naturally failed trajectories also creates credit-assignment ambiguity, since failed traces often mix benign mistakes with genuinely task-dooming actions. To address this, we propose *adaptive fault injection*. Starting from successful traces, we inject targeted perturbations at selected intermediate steps. A multi-armed bandit prioritizes error categories likely to produce high-risk prefixes, and rollout-based verification labels whether continuation from the perturbed prefix leads to downstream failure. This yields step-localized supervision focused on

[*]Equal contribution [1]Fujitsu Research [2]Carnegie Mellon University. Correspondence to: Vartika Sengar <vartika.sengar@fujitsu.com>.

Accepted at ICML 2026 Workshop: Forecasting as a New Frontier of Intelligence, Seoul, South Korea. Copyright 2026 by the author(s).

future recoverability rather than local action correctness. To reduce synthetic-artifact concerns, perturbations are derived from real failure categories, labels are assigned only after rollout verification, and evaluation includes human-annotated traces outside our generation pipeline.

Empirically, online failure forecasting is difficult. Frontier LLM forecasters remain limited: Claude-4.5 Sonnet achieves 59.87% weighted F1 at \$17.32 per test set, while GPT-4o achieves 68.66% at \$9.94. A lightweight Qwen-3-8B model fine-tuned only on our synthetic data achieves 78.01% weighted F1 on the same human-annotated benchmark (Zhang et al., 2025c), outperforming frontier models at \$0.13, roughly 100× lower inference cost. When used as a runtime monitor for selective intervention, the forecaster improves agent success rates across diverse tasks: HotpotQA (Yang et al., 2018) from 70% to 74%, MuSiQue (Trivedi et al., 2022) from 12% to 24%, Who&When from GAIA and AssistantBench (Zhang et al., 2025c) from 10% to 14%, SWE-Bench (Jimenez et al., 2023) from 67% to 69%, MBPP (Austin et al., 2021) from 47% to 49%, and EnterpriseBench (Vishwakarma et al., 2025) from 66% to 82%.

We summarize our main contributions as follows:

- We formalize *Task Failure Forecasting* as online prediction of policy-conditioned high-risk states from partial agent traces, distinct from post-hoc fault attribution and local process supervision.
- We propose *adaptive fault injection*: combine bandit-guided perturbation with rollout-based verification to generate step-level supervision targeting future recoverability rather than local action correctness.
- We show that a lightweight Qwen-3-8B forecaster trained on our synthetic data outperforms frontier LLMs on human-annotated benchmarks at approximately 100× lower cost, & enables runtime intervention that improves agent success rates across webQA, coding, & enterprise-agent tasks.

2. Problem Formulation

Let an agentic system be defined by a policy π interacting with an environment. We represent an execution trace as an alternating sequence of observable contexts and actions, $\tau = (o_0, a_0, o_1, a_1, \ldots, o_T, a_T)$, where o_0 is the task instruction, and $o_t \in \mathcal{O}$ denotes the observable context available at step t, including environment feedback, tool outputs, messages from other agents, and any intermediate agent-generated text exposed in the trace. Let $h_t = (o_0, a_0, \ldots, o_t)$ denote the pre-action history at step t, and let the agent choose an action $a_t \sim \pi(\cdot \mid h_t)$. We write the executed prefix up to step t as $\tau_{\leq t} = (o_0, a_0, \ldots, o_t, a_t)$. The forecaster receives this prefix $\tau_{\leq t}$ and predicts the risk of terminal task failure under continued execution. Because LLM-agent outcomes generally depend on the full interaction history rather than a compact Markov state, we use the executed prefix itself as the state representation and define $s_t \triangleq \tau_{\leq t}$. Forecasting is defined with respect to a fixed policy π, so recoverability is policy-conditioned rather than absolute property of a prefix. We define a binary terminal outcome function $Y(\tau) \in \{0, 1\}$, where $Y(\tau) = 1$ denotes task failure and $Y(\tau) = 0$ denotes task success. The *Task Failure Forecasting* problem is to learn a predictor

$$f_\theta(\tau_{\leq t}) := P(Y(\tau) = 1 \mid \tau_{\leq t}, \pi)\,, \tag{1}$$

which estimates probability that ongoing trajectory will eventually fail, conditioned on the currently executed prefix under policy π. This can also be viewed as value function estimation with sparse terminal rewards. Specifically, define reward sequence such that $r_k = 0$ for all $k < T$ and $r_T = Y(\tau)$. Then return from any partial prefix is exactly the final failure indicator. Accordingly, forecaster can be interpreted as approximating policy-conditioned *failure-value* function $F^\pi(s_t) := \mathbb{E}_{\tau\sim\pi}\left[\sum_{k=t}^{T} r_k \,\middle|\, s_t\right] = \mathbb{E}_{\tau\sim\pi}[Y(\tau) \mid s_t] = P(Y(\tau) = 1 \mid s_t, \pi)$. This connection motivates formulation, but our contribution is not a new RL value estimator; rather, we study how to obtain practical supervision for long-horizon LLM-agent traces where direct Monte Carlo estimation over many prefixes is prohibitively expensive.

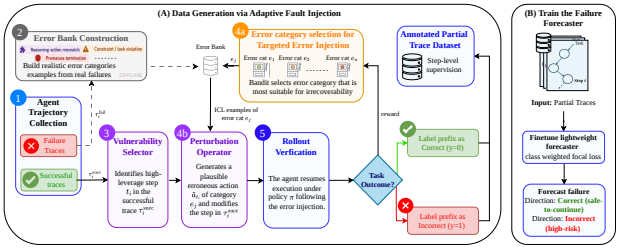


Figure 1. **Adaptive fault injection for partial-trace supervision.** Successful trajectories provide base traces, while failed trajectories are used offline to build an error bank of realistic failure categories and examples. A vulnerability selector identifies high-leverage steps in a successful trace. A bandit selects an error category likely to induce high-risk failure, and a perturbation operator injects a plausible faulty action. The agent then rolls out from the perturbed prefix under policy π, yielding a label: safe-to-continue ($y = 0$) if the rollout succeeds, and high-risk ($y = 1$) if it fails. The resulting partial traces train the failure forecaster.

3. Methodology

To train a failure forecaster from partial traces, we need labels that connect a trace prefix to its eventual task outcome. Naturally failed traces provide this supervision only weakly: they may contain multiple deviations, some recoverable and others task-dooming, so the terminal outcome alone does not reveal which step caused the loss of recoverability. Let $\mathcal{R}(\tau_{\leq t}; \pi)$ denote a stochastic rollout that completes a partial trace $\tau_{\leq t}$ using policy π. An error is **recoverable** if the policy can still succeed, i.e., $P(Y(\mathcal{R}(\tau_{\leq t}; \pi)) = 0) > 0$, and **irrecoverable** if this success probability is zero. Since

exact irrecoverability cannot be verified with finite rollouts, we use **high-risk** to refer to prefixes whose success probability under π is close to zero. For example, repeating a command such as `$ ls -la` is usually recoverable, while prematurely terminating before satisfying the user request is high-risk. We propose *adaptive fault injection* to generate supervision focused on the boundary between recoverable and high-risk states. Starting from successful traces, we inject plausible faults at selected intermediate steps and evaluate the resulting continuation through rollouts. This yields step-level stochastic labels that better connect a specific action to downstream failure. subsection 3.1 describes data generation and subsection 3.2 describes model training.

3.1. Data Generation via Adaptive Fault Injection

We synthesize a dataset with stochastic root-cause labels by injecting plausible faults into successful traces, isolating the effect of a single perturbation and verifying its downstream impact through rollouts.

(i) Error Bank Construction. To make injected faults realistic, we construct an *Error Bank* $\mathcal{B}$ from common LLM-agent failure categories (Cemri et al., 2025), such as "Reasoning-Action Mismatch", "Constraint Violation", etc. Given these categories and failed traces, an LLM extracts representative examples for each error type, which are later used to guide perturbation.

(ii) Fault Injection. For each successful trace, an LLM-based *Vulnerability Selector* $\mathcal{M}_{\text{sel}}$ identifies high-leverage steps, such as key reasoning steps or API calls. For a selected step t, a Multi-Armed Bandit policy π_{bandit} chooses an error category c from $\mathcal{B}$. The bandit follows an ϵ-greedy strategy (analyzed in RQ3 in section 11), and receives reward $r = 1$ if the perturbed rollout fails and $r = 0$ otherwise, allowing it to prioritize error types that more often induce high-risk prefixes. A perturbation operator LLM $\Pi(\cdot)$ then generates a plausible erroneous action $\tilde{a}_t$ conditioned on the selected category c and pre-action history h_t. This adaptive sampling concentrates compute on informative high-risk examples rather than uniformly sampling sparse failures. We treat the resulting dataset $\mathcal{D}$ as drawn from the bandit-induced hard-error distribution $\mathcal{P}_{\text{hard}}$.

(iii) Rollout-Based Stochastic Labeling. Exact failure probability would require infinitely many rollouts. Instead, we use single Monte Carlo rollout as a stochastic label (discussed in section 8). Let $y_{\text{roll}} \in \{0, 1\}$ denote whether rollout from $\tau_{\leq t}$ fails. Then $\mathbb{E}[y_{\text{roll}} \mid \tau_{\leq t}] = P(Y(\tau) = 1 \mid \tau_{\leq t}, \pi)$. Thus, each rollout provides noisy but unbiased sample of the policy-conditioned failure outcome. We use same capable policy π that generated the original successful traces, ensuring that labels reflect the policy being monitored. Full procedure in Algorithm 1.

3.2. Training

We train a lightweight binary forecaster f_θ to classify partial trace $\tau_{\leq t}$ as either *Safe-to-continue* $(y = 0)$ or *High-Risk* $(y = 1)$. Safe class includes both correct/no-error prefixes and recoverable-error prefixes, while high-risk class contains prefixes whose rollout fails under π. In this work, we fine-tune Qwen3-8B-Instruct (Yang et al., 2025) as f_θ.

Partial Trace Dataset Construction. For each failure-inducing trace with root-cause step t_{cause}, we label all prefixes before the root cause as safe, $\{\tau_{\leq t}, y = 0\}$ for $t < t_{\text{cause}}$, and the root-cause prefix as high-risk, $\{\tau_{\leq t_{\text{cause}}}, y = 1\}$. The dataset therefore contains three semantic sample types: correct/no-error prefixes, recoverable-error prefixes, and high-risk prefixes. The first two are mapped to $y = 0$, and the last to $y = 1$.

Training Objective. Because high-risk errors are rare, we train with class-weighted focal loss (Lin et al., 2017): $\ell_{\text{focal}} = -w_i(1 - p_t)^\gamma \log(p_t)$, where p_t is the predicted probability of the true class, γ is the focusing parameter, and w_i is an inverse-frequency weight based on sample type: $w_i = \beta \cdot \frac{\text{freq}_{\text{all}}}{\text{freq}_i}$. Here, freq_i is the frequency of the sample's type, freq_{all} is the mean frequency across types, and β is tuned on validation data. Hyperparameters and implementation details are provided in section 7.

4. Experimental Setup

Public Dataset. We evaluate on *Who and When* benchmark (Zhang et al., 2025c), which contains agent traces from GAIA (Mialon et al., 2023) and AssistantBench (Yoran et al., 2024). The benchmark includes 58 failed queries, 30 from GAIA and 28 from AssistantBench, produced by Magentic-One (Fourney et al., 2024) with GPT-4o. Expert annotators identify earliest irrecoverable error, or root-cause step t_{cause}. We convert these annotations into binary labels: $\tau_{\leq t_{\text{cause}}}$ is labeled High-Risk $(y = 1)$, and all earlier prefixes $\tau_{\leq t}$ for $t < t_{\text{cause}}$ are labeled Safe-to-continue $(y = 0)$.

Synthetic Dataset. Table 2 summarizes the data generated using section 3. Because recoverability is policy-conditioned, we construct each dataset with the corresponding task policy. For **WebQA**, we use Magentic-One (Fourney et al., 2024) with `gpt-4o`, using GAIA queries excluding *Who and When*, plus 1000 HotpotQA queries. For **Coding**, we use MBPP traces generated by Magentic-One with `gpt-4o`, using the MBPP training split for data construction and the validation split for downstream evaluation. For **Agentic** tasks, we follow EnterpriseBench and use a ReAct agent with

Category	Split	Total	Safe	Risky
WebQA	Train	3958	3562	396
	Test	480	430	50
	Test*	937	879	58
Coding	Train	3248	2816	432
	Test	630	583	47
Agentic	Train	1604	1251	353
	Test	410	315	95

Table 2. Sample distribution across training and test sets. *Safe* includes no-error and recoverable-error prefixes; *Risky* includes prefixes whose continuation fails. Test* - (Zhang et al., 2025c)

Model	Who&When (Zhang et al., 2025c) GAIA + AssistantBench				Our test GAIA + HotpotQA				Our test EnterpriseBench				Our test MBPP			
	Safe	Risky	W-Avg	Cost	Safe	Risky	W-Avg	Cost	Safe	Risky	W-Avg	Cost	Safe	Risky	W-Avg	Cost
Claude-4.5 Sonnet	62.9	16.0	59.9	$17.32	40.5	15.9	38.4	$3.73	68.3	**44.8**	62.9	$0.44	67.0	24.0	64.0	$0.74
Gemini-2.5 Pro	69.5	**17.7**	66.1	$5.93	47.1	16.9	44.4	$1.34	67.6	41.6	61.5	$0.11	66.0	**33.0**	63.0	$0.59
GPT-4o	72.4	14.9	68.7	$9.94	35.6	17.2	34.0	$2.26	48.7	37.0	46.0	$0.17	66.0	27.0	62.0	$0.59
Qwen-3 8B (Δ_b)	8.6	12.0	8.8	**$0.13**	25.4	16.0	24.5	**$0.03**	66.4	40.5	60.4	**$0.03**	29.0	19.0	28.0	**$0.01**
Ours (Δ_f)	**82.3**	16.2	**78.0**	**$0.13**	**67.8**	**19.7**	**63.6**	**$0.03**	**77.0**	42.1	**68.9**	**$0.03**	**72.0**	29.0	**68.0**	**$0.01**
$\Delta_f - \Delta_b$ [95% CI]	+73.7 [+70.6, +76.8]	+4.22 [+0.1, +8.5]	–	–	+52.8 [+47.2, +58.5]	+2.28 [+1.0, +5.5]	–	–	+10.6 [+7.2, +14.0]	+1.52 [-0.6, +6.0]	–	–	+38.2 [+31.7, +45.0]	+8.2 [+5.4, +11.2]	–	–

Table 1. **Failure forecasting F1 scores and inference costs.** "Safe" = Safe-to-continue F1; "Risky" = High-Risk F1; "W-Avg" = weighted F1. The shaded row reports gains over base Qwen-3 8B with 95% bootstrap confidence intervals and mean difference. Cost denotes total forecasting inference cost in USD. Bold marks best and underline marks second-best.

`GPT-5.1`, using 450 tasks for training-data construction and 50 held-out tasks for testing.

Baselines. We compare against LLMs prompted as online monitors: Gemini-2.5 Pro, Claude-4.5 Sonnet, GPT-4o, and Qwen-3-8B. At each step t, each model observes the partial trace $\tau_{\leq t}$ and predicts whether continuation will fail. Our method fine-tunes Qwen-3-8B only on the synthetic training data for the corresponding task category.

Metrics. We report step-level binary classification performance using weighted F1 over Safe-to-continue and High-Risk classes. We also report inference cost for evaluating the full test set, computed using standard per-1M-token API prices from `OpenRouter`.

5. Experimental Results

We evaluate our failure forecasting framework along four research dimensions:

- **Forecasting Capability:** We benchmark accuracy against general-purpose LLMs (**RQ1**) and test robustness across agent policies (**RQ6**, section 11).
- **Downstream Utility:** We use forecaster for inference-time scaling and measure its effect on agent success across held-out and unseen tasks (**RQ2**).
- **Data Generation Validity:** We evaluate adaptive fault injection efficiency (**RQ3**, section 11) and generated-trace plausibility (**RQ4**, section 11).
- **Problem Characterization:** We study forecasting difficulty via human judgments and by embedding separability of partial traces (**RQ5**, section 11).

> ***RQ1:*** *How accurately and efficiently can the forecaster detect high-risk errors on human-annotated and held-out synthetic traces?*

Table 1 compares our fine-tuned Qwen-3-8B forecaster with zero-shot LLM monitors. On the human-annotated Who&When benchmark, our model achieves the best weighted F1, 78.01, outperforming GPT-4o, 68.66, and base Qwen-3-8B, 8.78. It also remains substantially cheaper, costing $0.13 per test set, approximately 100× lower than Claude-4.5 Sonnet. Similar gains hold across our held-out WebQA, EnterpriseBench, and MBPP test sets.

Takeaway: *Synthetic supervision yields a forecaster that generalizes to human-labeled failures and is much cheaper than frontier LLM monitors.*

> ***RQ2:*** *Can the forecaster selectively allocate inference-time scaling compute to improve task success?*

Dataset	Forecaster/Strategy	Accuracy	Cost
Short-trace tasks *(10–30 steps)*			
HotpotQA (Held-out)	None	0.70	$27.50
	Claude 4.5 Sonnet + BoN	0.72	$39.80
	Qwen-3 8B + BoN	0.66	$30.00
	Ours + BoN	0.74 ↑5.7%	$30.01
MuSiQue (Unseen)	None	0.12	$20.33
	Claude 4.5 Sonnet + BoN	0.14	$35.68
	Qwen-3 8B + BoN	0.10	$22.31
	Ours + BoN	0.24 ↑100.0%	$22.35
EnterpriseBench (Held-out)	None	0.66	$4.57
	Claude 4.5 Sonnet + BoN	0.84	$9.23
	Qwen-3 8B + BoN	0.78	$7.86
	Ours + BoN	0.82 ↑24.2%	$7.83
Who&When (Zhang et al., 2025c)	None	0.10	$32.57
	Claude 4.5 Sonnet + BoN	0.12	$53.65
	Qwen-3 8B + BoN	0.09	$40.80
	Ours + BoN	0.14 ↑40.0%	$35.63
MBPP (Held-out)	None	0.47	$3.13
	Claude 4.5 Sonnet + BoN	0.51	$9.28
	Qwen-3 8B + BoN	0.47	$7.56
	Ours + BoN	0.49 ↑4.3%	$7.18
Long-trace tasks *(60–65 steps)*			
SWE-Bench (Held-out)	None	0.67	$8.48
	Claude 4.5 Sonnet + Self-Refine	0.70	$214.90
	Qwen-3 8B + Self-Refine	0.64	$8.82
	Ours + Self-Refine	0.69 ↑3.0%	$10.17

Table 3. **Inference scaling performance.** Costs include base execution, forecasting, and triggered scaling.

We use the forecaster as a gate for inference-time scaling. At each step, the agent proposes an action $a_t \sim \pi(\cdot \mid \tau_{<t})$, and the forecaster evaluates $\tau_{\leq t}$. If failure is predicted and budget remains, we discard a_t and invoke a scaling module $\mathcal{M}_{\text{scaling}}$ to generate an alternative action. Each query receives budget $K = 5$ interventions. For short-trace tasks, we use step-level Best-of-N with $N = 3$. For SWE-Bench, whose OpenHands (Wang et al., 2025) traces are

longer and may require revising earlier decisions, we use Self-Refine instead. As shown in Table 3, our gate improves accuracy on all tasks

Takeaway: *The forecaster enables targeted intervention, improving downstream success while avoiding the cost of always using large LLM gates.*

References

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization methods for large-scale machine learning. *SIAM review*, 60(2):223–311, 2018.

Cemri, M., Pan, M. Z., Yang, S., Agrawal, L. A., Chopra, B., Tiwari, R., Keutzer, K., Parameswaran, A., Klein, D., Ramchandran, K., et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

Deshpande, D., Gangal, V., Mehta, H., Krishnan, J., Kannappan, A., and Qian, R. Trail: Trace reasoning and agentic issue localization. *arXiv preprint arXiv:2505.08638*, 2025.

Fourney, A., Bansal, G., Mozannar, H., Tan, C., Salinas, E., Niedtner, F., Proebsting, G., Bassman, G., Gerrits, J., Alber, J., et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.

Hossain, M. S., Betts, J. M., and Paplinski, A. P. Dual focal loss to address class imbalance in semantic segmentation. *Neurocomputing*, 462:69–87, 2021.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. R. Swe-bench: Can language models resolve real-world github issues? In *The twelfth international conference on learning representations*, 2023.

Kong, F., Zhang, R., Yin, H., Zhang, G., Zhang, X., Chen, Z., Zhang, Z., Zhang, X., Zhu, S.-C., and Feng, X. Aegis: Automated error generation and attribution for multi-agent systems. *arXiv preprint arXiv:2509.14295*, 2025.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

Lin, S., Hilton, J., and Evans, O. Teaching models to express their uncertainty in words. *arXiv preprint arXiv:2205.14334*, 2022.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

Luo, L., Liu, Y., Liu, R., Phatale, S., Guo, M., Lara, H., Li, Y., Shu, L., Zhu, Y., Meng, L., et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

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

Mohri, M., Rostamizadeh, A., and Talwalkar, A. *Foundations of machine learning*. MIT press, 2018.

Robbins, H. and Monro, S. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Saunders, W., Yeh, C., Wu, J., Bills, S., Ouyang, L., Ward, J., and Leike, J. Self-critiquing models for assisting human evaluators. *arXiv preprint arXiv:2206.05802*, 2022.

Shen, Y., Song, K., Tan, X., Li, D., Lu, W., and Zhuang, Y. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Sutton, R. S., Barto, A. G., et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Trivedi, H., Balasubramanian, N., Khot, T., and Sabharwal, A. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Uesato, J., Kushman, N., Kumar, R., Song, F., Siegel, N., Wang, L., Creswell, A., Irving, G., and Higgins, I. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Vishwakarma, H., Agarwal, A., Patil, O., Devaguptapu, C., and Chandran, M. Can llms help you at work? a sandbox for evaluating llm agents in enterprise environments. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 9178–9212, 2025.

Wang, P., Li, L., Shao, Z., Xu, R., Dai, D., Li, Y., Chen, D., Wu, Y., and Sui, Z. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., et al. Openhands: An open platform for ai software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W., Salakhutdinov, R., and Manning, C. D. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Yoran, O., Amouyal, S., Malaviya, C., Bogin, B., Press, O., and Berant, J. Assistantbench: Can web agents solve realistic and time-consuming tasks? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8938–8968, 2024.

Zhang, G., Wang, J., Chen, J., Zhou, W., Wang, K., and Yan, S. Agentracer: Who is inducing failure in the llm agentic systems? *arXiv preprint arXiv:2509.03312*, 2025a.

Zhang, K., Zhang, J., Li, H., Zhu, X., Hua, E., Lv, X., Ding, N., Qi, B., and Zhou, B. Openprm: Building open-domain process-based reward models with preference trees. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Zhang, S., Yin, M., Zhang, J., Liu, J., Han, Z., Zhang, J., Li, B., Wang, C., Wang, H., Chen, Y., et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. In *Forty-second International Conference on Machine Learning*, 2025c.

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

Zhu, C., Hong, S., Wu, J., Chawla, K., Tang, C., Yin, Y., Wolfe, N., Babinsky, E., and Liu, D. Raffles: Reasoning-based attribution of faults for llm systems. *arXiv preprint arXiv:2509.06822*, 2025.

Appendix

In this section, we provide additional results and details that we could not include in the main paper due to space constraints. We will open source the code upon paper acceptance. In particular, this appendix contains the following:

- **Proposed Algorithm**: Complete algorithmic description.
- **Forecasting Model Training Details**: Hyperparameters & implementation specifics for the forecasting model.
- **Analysis: Compute-Coverage Tradeoff**: Formalizes the fixed-budget tradeoff between labeling many diverse states and estimating fewer states more precisely, showing why broad state coverage can be preferable for long-horizon failure forecasting.
- **Proposition Proof and Discussion**: Formal proof of the risk decomposition and extended discussion of the compute-coverage tradeoff.
- **Related Works**: Comprehensive review of prior work on agent failures, process supervision, and value estimation.
- **Additional Results**: Continuation of research questions from section 5, including bandit allocation analysis, trace quality evaluation, human annotation study, embedding analysis, and policy generalization experiments.
- **Limitation and Future Works**: This section discuss the limitation and future directions.
- **Prompts**: Complete prompt templates used for fault analysis, vulnerability selection, attack generation, and utility evaluation.
- **Human Evaluation Details**: Annotation interface screenshots & detailed methodology for the studies reported in the main paper.

6. Proposed Algorithm

Algorithm1 describes our adaptive fault injection procedure for generating high-fidelity supervision for task failure forecasting, as detailed in section 3 of the main paper. Starting from a successful execution trace, the algorithm first identifies a subset of vulnerable steps using a selector LLM that highlights high-leverage decision points. For each selected step, an error category is sampled from an adaptive multi-armed bandit, which prioritizes fault types that have historically led to high-risk failures. A perturbed action consistent with the chosen error category is then synthesized by perturbation LLM and injected into the trace, after which the agent policy performs a rollout to completion. The outcome of this rollout serves as a stochastic verification signal: if continuation from the perturbed trace fails, the injected step is labeled as a high-risk transition point and added to the dataset, and the bandit receives a positive reward; otherwise, the bandit is updated with zero reward. By iterating this process, the algorithm efficiently concentrates computation on high-yield error types and produces step-level labels.

Algorithm 1 Data Generation via Adaptive Fault Injection

Require: Successful trace $\tau^* = (x_0, a_0^*, \ldots, a_T^*)$, Error Bank $\mathcal{B}$, Bandit Policy π_{bandit}, Selector Model $\mathcal{M}_{\text{sel}}$.
Initialize dataset $\mathcal{D} \leftarrow \emptyset$
// 1. Identify vulnerable steps in successful trace
$\mathcal{T}_{\text{targets}} \leftarrow \mathcal{M}_{\text{sel}}(\tau^*)$ {Returns list of indices $\{t_1, t_2, \ldots\}$ suitable for attack}
for $t \in \mathcal{T}_{\text{targets}}$ **do**
  // 2. Select error type & generate perturbation
  Sample error category $c_t \sim \pi_{\text{bandit}}$
  Retrieve eg. $E \leftarrow \text{GetExamples}(\mathcal{B}, c_t)$
  Generate perturbed action $\tilde{a}_t \leftarrow \text{LLM}(a_t^*, c_t, E)$
  // 3. Verify high-risk failure via rollout
  // Single rollout for stochastic verification
  $\tau_{\text{roll}} \leftarrow \text{Rollout}(\tau^*_{<t} \cup \{\tilde{a}_t\}; \pi)$
  // check whether the perturbed continuation fails
  if $Y(\tau_{\text{roll}}) = 1$ **then**
    $t_{\text{cause}} \leftarrow t$
    Add $(\tau_{\text{roll}}, t_{\text{cause}})$ to $\mathcal{D}$
    Update π_{bandit} with reward $r = 1$ (Failure-inducing perturbation)
  else
    Update π_{bandit} with reward $r = 0$
  end if
end for
return Dataset $\mathcal{D}$

7. Forecasting Model Training Details

In this section, we continue the discussion of the training procedure introduced in subsection 3.2 and provide detailed implementation specifics for the forecasting model.

We fine-tuned a Qwen-3 8B language model using supervised fine-tuning (SFT) on partial execution traces for failure forecasting. The dataset exhibits a pronounced class imbalance across three outcome categories: *high-risk*, *recoverable*, and *correct*, with frequencies of 10%, 27%, and 63%, respectively.

To mitigate the effects of class imbalance, we employed a weighted loss formulation combined with focal loss. Class weights were initially derived using inverse-frequency scaling and subsequently multiplied by a constant scaling fac-

tor. This scaling was introduced to mitigate the vanishing gradient problem commonly observed when using focal loss(Hossain et al., 2021). This procedure yielded approximate effective weight ratios of 140:60:20 for high-risk, recoverable, and correct classes. However, ablation studies on the validation set (a small subset of train set) revealed that a modified weighting scheme of 140:40:20 resulted in superior performance. We additionally set the focal loss focusing parameter to $\gamma = 5$, which improved robustness to easy examples and enhanced discrimination among hard failure cases.

The constant scaling factor β was selected via systematic validation ablations. Specifically, we swept β over the range $[100, 300]$ in increments of 20 on the validation set. Based on these experiments, we set $\beta = 220$, which consistently yielded the best validation performance.

Training was conducted using Low-Rank Adaptation (LoRA) (Hu et al., 2022) with rank $r = 128$ and scaling factor $\alpha = 32$. Fine-tuning was performed on four NVIDIA A40 GPUs for four epochs, using an effective batch size of 160. The total training time was approximately eight hours.

Overall, this configuration provided a strong balance between training stability and predictive performance under severe class imbalance.

8. Analysis: Compute-Coverage Tradeoff

Here, we provide a theoretical analysis of our data generation strategy with respect to training a forecasting model on the generated data. Purpose of this analysis is to make explicit tradeoff between label precision and state coverage under a fixed rollout budget. Our analysis studies a simplified surrogate objective rather than the exact focal-loss objective used in training. Proof in section 9.

Proposition 8.1 (Simplified Risk Decomposition)**.** *Let $y^*(s) := P(Y(\tau) = 1 \mid s, \pi)$ denote the policy-conditioned failure probability at state s, and let $\mathcal{E}(f_\theta) = \mathbb{E}_{s\sim\mathcal{P}_{hard}}[\ell(f_\theta(s), y^*(s))]$ be the true expected risk of the forecasting model under a bounded surrogate loss ℓ. Now, suppose we have access to a dataset $\mathcal{D} \sim \mathcal{P}_{hard}$ containing S distinct states, where labels are estimated using N rollout outcomes, subject to a fixed compute budget $B = S \cdot N$. Then, under the assumptions stated in section 9, the risk is bounded with high probability as:*

$$\mathcal{E}(f_\theta) \leq \hat{R}_S(f_\theta) + \underbrace{\mathcal{O}\left(\frac{1}{\sqrt{S}}\right)}_{\textit{Coverage / Generalization Term}} + \underbrace{\mathcal{O}\left(\frac{1}{\sqrt{N}}\right)}_{\textit{Rollout-Noise Term}}$$

where $\hat{R}_S(f_\theta) = \frac{1}{S}\sum_{i=1}^{S} \ell(f(x_i), \tilde{y}_i)$ represents empirical surrogate risk on constructed dataset.

Implications for Data Generation. Under fixed budget B, substituting $S = B/N$ yields total error scaling as $\mathcal{O}\left(\sqrt{\frac{N}{B}}\right) + \mathcal{O}\left(\frac{1}{\sqrt{N}}\right)$. This reveals two extreme allocation strategies.

High Precision ($N \gg 1$): this reduces rollout noise but limits state diversity (coverage), causing the coverage term to scale less favorably as $\mathcal{O}(\sqrt{N/B})$. *High Diversity* ($N = 1$): this maximizes the number of labeled states ($S = B$) at cost of noisier labels, with coverage term scaling as $\mathcal{O}(1/\sqrt{B})$.

The single-rollout setting also admits a simple stochastic-optimization interpretation. Let y be a rollout-derived binary label sampled from the policy-conditioned failure process. For a differentiable surrogate loss, we have $\mathbb{E}_y[\nabla_\theta \ell(f_\theta(\tau_{\leq t}), y)] = \nabla_\theta \mathbb{E}_y[\ell(f_\theta(\tau_{\leq t}), y)]$, ensuring SGD convergence under standard conditions (Robbins & Monro, 1951; Bottou et al., 2018). That is, it does not introduce bias, although they do increase variance. This observation clarifies why single-rollout supervision can still be effective when broad state coverage matters more than precise per-state estimation.

Finally, this proposition's role is to make explicit a tradeoff: when limited state coverage is dominant source of error (our case as we deal with long-horizon agent traces, where the effective state space is large), it can be preferable to label more distinct states once rather than estimate a small number of states very precisely. To empirically study benefits of state coverage, we test under iso-compute conditions in Table 13 (section 11), and find that it indeed helps to allocate rollout budget toward labeling a larger number of distinct states rather than estimating a fewer states more precisely.

9. Proposition Proof and Discussion

Here we provide the complete proof of Proposition 1 from section 8, which justifies our single-rollout data generation strategy through a formal risk decomposition. This analysis establishes the theoretical foundation for prioritizing state coverage over label precision under fixed computational budgets.

Proof Setup. We treat the bandit sampler as a data generating process defining a distribution $\mathcal{P}_{\text{hard}}$. The bandit is run offline to produce a dataset $\mathcal{D} = \{(x_i, \tilde{y}_i)\}_{i=1}^{S}$ of size S, where labels $\tilde{y}_i$ are estimated via N rollouts. Learning proceeds via empirical risk minimization on $\mathcal{D}$. The following analysis characterizes the upper bound on the true risk with respect to $\mathcal{P}_{\text{hard}}$.

Proof. Let $\mathcal{H}$ be the hypothesis class of our model f_θ. Let $y^*(x) = \Pr(Y = 1|x)$ denote the true failure probability of a state x. Our data generation process produces a noisy label $\tilde{y}(x) = \frac{1}{N}\sum_{k=1}^{N} z_k$, where $z_k \sim \text{Bernoulli}(y^*(x))$,

representing the average of N rollouts. We assume the loss function $\ell(\hat{y}, y)$ used to train f_θ is L-Lipschitz and bounded in $[0, 1]$. i.e., we assume $|\ell(\hat{y}, y) - \ell(\hat{y}, y')| \leq L|y - y'|$ for any prediction $\hat{y}$ and $y, y' \in [0, 1]$. We discuss the validity of the Lipschitz assumption for deep neural networks with cross-entropy loss. If the input space (state space) for x is compact and the activation functions are continuous, then the logits are bounded due to this continuity. Consequently, the prediction probabilities computed using a softmax function remain distanced from the boundary of the probability simplex. This ensures that the Lipschitz condition mentioned above holds for cross-entropy loss.

We distinguish between three quantities:

1. **True Risk** $\mathcal{E}(f) = \mathbb{E}_{x \sim \mathcal{P}_{\text{hard}}}[\ell(f(x), y^*(x))]$.
2. **Noisy Population Risk** $R_{\text{noisy}}(f) = \mathbb{E}_{x,\tilde{y}}[\ell(f(x), \tilde{y}(x))]$.
3. **Empirical Risk** $\hat{R}_S(f) = \frac{1}{S}\sum_{i=1}^{S} \ell(f(x_i), \tilde{y}_i)$.

I. Error Decomposition. By the triangle inequality, we bound the total error (difference between True Risk and Empirical Risk):

$$\mathcal{E}(f) \leq \hat{R}_S(f) + \underbrace{|\mathcal{E}(f) - R_{\text{noisy}}(f)|}_{\text{(A) Label Noise Gap}} + \underbrace{|R_{\text{noisy}}(f) - \hat{R}_S(f)|}_{\text{(B) Generalization Gap}}$$

II: Bounding Label Noise (A). This term captures the deviation caused by training on finite rollouts $\tilde{y}$ rather than true probabilities y^*. Using the Lipschitz property:

$$|\mathcal{E}(f) - R_{\text{noisy}}(f)| \leq L \cdot \mathbb{E}_{x,\tilde{y}}[|y^*(x) - \tilde{y}(x)|]$$

By Jensen's inequality ($\mathbb{E}[|X|] \leq \sqrt{\mathbb{E}[X^2]}$), we bound the expected absolute deviation by the standard deviation of the estimator. Since $\tilde{y}$ is the mean of N i.i.d. Bernoulli variables:

$$\mathbb{E}[|y^*(x) - \tilde{y}(x)|] \leq \sqrt{\text{Var}(\tilde{y})} \leq \frac{1}{2\sqrt{N}}$$

III: Bounding Generalization (B). This term represents the standard uniform convergence gap. By applying standard Rademacher complexity bounds (Mohri et al., 2018), with probability at least $1 - \delta$:

$$\sup_{f \in \mathcal{H}} |R_{\text{noisy}}(f) - \hat{R}_S(f)| \leq 2\mathfrak{R}_S(\mathcal{H}) + 3\sqrt{\frac{\log(2/\delta)}{2S}}$$

Now, by substituting (II) and (III) back into (I), we obtain bound presented in Proposition 1:

$$\mathcal{E}(f) \leq \hat{R}_S(f) + \underbrace{2\mathfrak{R}_S(\mathcal{H}) + 3\sqrt{\frac{\log(2/\delta)}{2S}}}_{\text{Generalization Term}} + \underbrace{\frac{L}{2\sqrt{N}}}_{\text{Label Noise Term}}$$

□

Discussion. To determine a suitable data generation strategy, we analyze how the error bound behaves under a fixed computational budget B. Assuming the training algorithm successfully minimizes the empirical risk $\hat{R}_S$, we focus on minimizing the remaining structural error terms. Substituting $S = B/N$ into the bound, the *total error gap* scales as: $\mathcal{E}_{\text{gap}} \sim \mathcal{O}\left(\sqrt{\frac{N}{B}}\right) + \mathcal{O}\left(\frac{1}{\sqrt{N}}\right)$. We compare two distinct allocation strategies based on this relationship:
Case 1: High Precision ($N \gg 1$). By averaging many rollouts per state, we minimize the label noise term ($\mathcal{O}(\frac{1}{\sqrt{N}})$). However, this forces a reduction in unique states S, causing the generalization error to scale unfavorably as $\mathcal{O}(\sqrt{N/B})$.
Case 2: High Diversity ($N = 1$). By using a single rollout per state, we maximize the number of unique states $S = B$. While this maximizes label noise, the generalization error is minimized, scaling as $\mathcal{O}(\sqrt{1/B})$.

Conclusion. We argue that *Case 2 (Diversity)* is the superior strategy for two reasons:
(i) *Dimensionality:* In the domain of LLM agent traces, the state space is high-dimensional and sparse. Thus, the model is far more likely to fail due to encountering out-of-distribution states (high $\mathcal{G}$) than due to noisy labels on known states.
(ii) *Unbiased Gradients:* The key observation underlying our use of a single rollout per perturbed state ($N = 1$) is that, while such labels are noisy, they are not biased - thus training using SGD converges. Specifically, a rollout outcome $y \sim P(\cdot \mid \tau_{\leq t}, \pi)$ is a Monte Carlo sample of the true failure indicator, satisfying $\mathbb{E}[y \mid \tau_{\leq t}] = P(Y = 1 \mid \tau_{\leq t}, \pi)$. When training the forecaster with a differentiable loss $\ell(f_\theta(\tau_{\leq t}), y)$ whose gradient has finite second moment, and using independently generated Monte Carlo rollout labels, the resulting stochastic gradient satisfies $\mathbb{E}_y[\nabla_\theta \ell(f_\theta(\tau_{\leq t}), y)] = \nabla_\theta \mathbb{E}_y[\ell(f_\theta(\tau_{\leq t}), y)]$. Under these conditions and standard Robbins–Monro step-size assumptions, classical stochastic approximation theory guarantees that stochastic gradient descent converges (in expectation) to a stationary point of the true expected risk (Robbins & Monro, 1951; Bottou et al., 2018).

10. Related Works

We position our work within broader landscape of LLM agent failures, process supervision, and outcome prediction. We clarify how Task Failure Forecasting addresses gaps left by existing approaches.

Prior studies have examined agent failures largely post-hoc, evaluated local reasoning quality via step-level supervision, or estimated outcomes using rollout-based value approximations, but each of these perspectives leaves open the question of when an ongoing agent trace has already become irrecoverable. We position *Task Failure Forecasting* as a distinct problem focused on predicting inevitable task failure from partial execution traces under a fixed policy, by shifting attention from post-hoc fault attribution or local correctness to online prediction of future success reachability.

LLM Agents and Failure Analysis: LLMs are increasingly used as multi-step autonomous agents (Yao et al., 2022; Shen et al., 2023), with benchmarks exposing frequent long-horizon failures (Mialon et al., 2023; Yoran et al., 2024). Prior work largely studies failures post-hoc, via human annotation or fault localization (Fourney et al., 2024; Zhang et al., 2025c), which is ambiguous due to cross-step dependencies and error propagation in agent interaction graphs (Cemri et al., 2025; Deshpande et al., 2025). Automated approaches such as AgentTracer (Zhang et al., 2025a) and AEGIS (Kong et al., 2025) attribute failures after task completion using counterfactual replay and fault injection. In contrast, we study *online* forecasting: predicting high-risk failure states from partial traces under a fixed policy.

Step-Level Judgments and Process Supervision: LLMs have been used as critics, verifiers, and process supervisors (Lin et al., 2022; Saunders et al., 2022; Shinn et al., 2023; Bai et al., 2022; Uesato et al., 2022; Lightman et al., 2023; Zhang et al., 2025b). These methods primarily evaluate local reasoning quality or preference consistency. However, recent analysis show that Monte Carlo based step labeling, where a base policy is used to continue reasoning from intermediate steps, can be unreliable when the policy is imperfect, mislabeling both correct and flawed steps (Zhang et al., 2025d). As a result, locally incorrect steps may be recoverable, while seemingly plausible steps may irreversibly constrain future outcomes. Task Failure Forecasting instead directly estimates future success reachability, a signal not captured by step-quality supervision alone.

Outcome-Based Supervision and Forecasting: Human trace annotation suffers from ambiguity, hindsight bias, and low inter-annotator agreement (Zhang et al., 2025c; Zhu et al., 2025), and often assumes a single failure-causing step in long-horizon, multi-agent traces. Outcome prediction is closely related to value estimation in reinforcement learning (Sutton et al., 1998), where Monte Carlo rollouts provide unbiased but high-variance estimates (Robbins & Monro, 1951). In language-based reasoning, rollout-based estimation is highly sensitive to policy quality, limiting its reliability in practice (Zhang et al., 2025d). We formalize Task Failure Forecasting as approximating the complement of the value function and use rollout-verified fault injection to obtain unambiguous supervision.

While prior work studies agent failures, self-critique, step supervision, and value estimation, none address forecasting *high-risk failure* from partial traces. We introduce Task Failure Forecasting as a distinct objective with adaptive fault injection and rollout-based supervision.

11. Additional Results

This section extends the experimental evaluation from section 5 with additional research questions and analysis that provide deeper insights into our method's performance and characteristics.

In section 5 of the main paper, we established that (1) our forecaster outperforms/performs-on-par with general-purpose LLMs on failure forecasting which being substantially cheaper, (2) enables effective inference-time scaling, and (3) benefits from adaptive fault injection. Here, we extend that analysis with four additional studies: a detailed breakdown of bandit allocation efficiency (continuation of RQ3), evaluation of synthetic trace plausibility (RQ4), characterization of forecasting task difficulty (RQ5), and assessment of generalization across agent policies (RQ6).

> ***RQ3:*** *Does adaptive fault injection improve the yield of valid high-risk training examples compared to random perturbation?*

We evaluate efficiency of our data generation pipeline via an ablation on perturbation-selection policy. We compare our ε-greedy Bandit selector ($\varepsilon = 0.1$), against uniform random selection and two standard bandit algorithms (Thompson Sampling and UCB). Our primary metric is *Attack Success Rate* (ASR) defined as the proportion of perturbed traces verified as high-risk ($y = 1$) via rollouts. To compare policies under fixed perturbation budget, we compute *cumulative ASR* at regular intervals, aggregating outcomes into bins to reduce noise and make trends comparable across policies. This analysis determines whether adaptive perturbation selection improves efficiency of generating high-risk examples and helps in maximizing density of task failure signals, ensuring that our computational budget is spent on generating valid training targets rather than benign/recoverable errors.

Across both GAIA and HotpotQA, ε-greedy consistently yields the highest ASR. On GAIA (Table 7a), ASR increases

Dataset	Component	# LLM Calls	Avg In Tokens	Avg Out Tokens	Cost ($)
MBPP with MagenticOne	Selector LLM	401	1228	227	2.14
	Injector LLM	1476	1,466	500	12.79
	Rollout Operation	6286	1,223	258	35.44
	Error Bank Creation	346	2198	470	3.52
	Total	**8,509**			**$53.89**
EnterpriseBench wth ReAct	Selector LLM	548	2029	601	$4.68
	Injector LLM	1753	10898	339	$29.82
	Rollout Operation	10342	4629	92	$69.36
	Error Bank Creation	209	8722	487	$3.30
	Total	**12852**			**$107.16**

Table 4. **Cost breakdown by pipeline component.** We report the number of LLM calls, average input and output tokens per call, and total inference cost for each component. The Bandit Controller is omitted because it does not use an LLM. For MBPP, `gpt-4o` was the LLM used for all components, and for EnterpriseBench it was `gpt-5.1`. We chose the same LLM that was used by the agents.

over time with ε-greedy outperforming others. On HotpotQA (Table 7b), ASR decreases across all methods as the perturbation space becomes harder to exploit; yet ε-greedy maintains highest ASR throughout. We attribute this to algorithm's strong exploitation of empirical means, which is useful for low-budget cases like ours (owing to expense of agent rollouts).

This efficiency is further quantified in Table 9 (section 11), where bandit allocates nearly 7× more attacks to high-yield categories (e.g., "Reasoning-Action Mismatch") while down-weighting unproductive ones (e.g., "Unaware of Term. Cond.") compared to random baseline. This efficiency translates directly to downstream forecasting performance. As shown in Table 8, models trained on bandit-selected perturbations achieve a weighted F1 of 78.01% on the "Who and When" benchmark, compared to 57.32% when training on randomly selected perturbations, an improvement attributable to better data curation.

Model	Class	F1 Score
Ours (w/ Bandit)	Safe-to-continue	**82.28**
	High-risk	**16.23**
	Weighted Avg	***78.01***
Ours (w/ Random)	Safe-to-continue	60.31
	High-Risk	14.00
	Weighted Avg	*57.32*

Table 8. **Classification Performance ("Who and When" benchmark).** Comparison of F1 scores across classes for our method using Bandit selection vs Random (under same perturbation budget).

Table 9 provides a fine-grained breakdown of how adaptive perturbation selection redistributes a fixed attack budget across failure categories for HotpotQA and GAIA. Categories are grouped by their baseline (random) ASR to highlight whether the Bandit policy reinforces or suppresses exploration relative to empirical yield. The results show that the Bandit systematically allocates more attacks to high-return categories (e.g., *Disobey task specification*) while sharply reducing exploration of consistently low-yield failures (e.g., *Unaware of Termination Condition*), even when those categories appear frequently under random sampling. This behavior illustrates that improvements in cumulative ASR arise primarily from better budget allocation rather than uniformly higher per-category success rates, validating the role of adaptive selection in improving data efficiency.

Takeaway: *Adaptive fault injection is more effective than random perturbation and significantly improves data efficiency and forecasting accuracy.*

RQ4: *Are the traces generated by our algorithm plausible and realistic?*

To ensure that our synthetic training data remains valid and does not introduce artificial artifacts that are trivial for a model to spot, we conducted a blinded human evaluation with expert annotators. We designed two complementary studies to assess both the plausibility of the traces and the fidelity of the injected faults. In Study 1, annotators were presented with a mix of unperturbed and perturbed traces and asked to rate them on three dimensions: (i) *Trace Coherence*: whether the sequence of states and actions is logically consistent up to the final step; (ii) *Action Plausibility*: whether the final action, even if erroneous, is a mistake a realistic agent might make (as opposed to gibberish) . In Study 2, we focused exclusively on the perturbed samples to evaluate *Semantic Alignment*, determining if the realized error linguistically aligns with the specific category selected by the Bandit policy (e.g., verifying that a "Constraint Violation" injection actually violated a constraint). We conducted the blinded human evaluation with two expert annotators over 100 traces (50 unperturbed, 50 perturbed) from our test set. For study two, all perturbed samples in the test set were used.

Takeaway: *Human evaluation in Table 10 shows that our data generation pipeline produces coherent, plausible traces with fault injections that faithfully reflect the intended error*

Model	Class	Precision	Recall	F1-Score	Fore. Cost
Claude-4.5 Sonnet	Safe-to-continue	96.49	46.67	62.91	$17.32
	High-Risk	8.92	75.44	15.96	
	Weighted Avg	*90.83*	*48.53*	*59.87*	
Gemini-2.5 Pro	Safe-to-continue	96.75	54.19	69.47	$5.93
	High-Risk	10.02	73.68	**17.65**	
	Weighted Avg	***91.13***	*55.45*	*66.11*	
Qwen-3 8B	Safe-to-continue	92.5	4.48	8.55	**$0.13**
	High-Risk	6.41	94.74	12.01	
	Weighted Avg	*86.94*	*10.32*	*8.78*	
GPT-4o	Safe-to-continue	95.06	58.44	72.39	$9.94
	High-Risk	8.56	56.14	14.85	
	Weighted Avg	*89.46*	*58.30*	*68.66*	
Ours	Safe-to-continue	94.93	**72.61**	**82.28**	**$0.13**
	High-Risk	9.96	43.86	16.23	
	Weighted Avg	*89.44*	***70.75***	***78.01***	

(a) Who and When (GAIA + AssistantBench) (Zhang et al., 2025c)

Model	Class	Precision	Recall	F1-Score	Fore. Cost
Claude-4.5 Sonnet	Safe-to-continue	91.57	26.03	40.53	$3.73
	High-Risk	8.86	75.00	15.85	
	Weighted Avg	*84.33*	*30.31*	*38.37*	
Gemini-2.5 Pro	Safe-to-continue	92.93	31.51	47.06	$1.34
	High-Risk	9.50	75.00	16.87	
	Weighted Avg	*85.63*	*35.31*	*44.42*	
Qwen-3 8B	Safe-to-continue	91.49	14.73	25.37	**$0.03**
	High-Risk	8.79	85.71	15.95	
	Weighted Avg	*84.25*	*20.94*	*24.54*	
GPT-4o	Safe-to-continue	**94.12**	21.92	35.56	$2.26
	High-Risk	9.56	85.71	17.20	
	Weighted Avg	*86.72*	*27.50*	*33.95*	
Ours	Safe-to-continue	**94.00**	**53.08**	**67.83**	**$0.03**
	High-Risk	**11.61**	64.29	**19.67**	
	Weighted Avg	***86.74***	***54.06***	***63.62***	

(b) Our test set (GAIA + HotpotQA)

Model	Class	Precision	Recall	F1-Score	Fore. Cost
Claude-4.5 Sonnet	Safe-to-continue	86.41	56.51	68.33	$0.44
	High-Risk	32.84	70.53	**44.82**	
	Weighted Avg	*74.00*	*59.76*	*62.88*	
Gemini-2.5 Pro	Safe-to-continue	83.96	56.51	67.55	$0.11
	High-Risk	30.81	64.21	41.63	
	Weighted Avg	*71.65*	*58.29*	*61.54*	
Qwen-3 8B	Safe-to-continue	83.25	55.24	66.41	**$0.03**
	High-Risk	29.85	63.16	40.54	
	Weighted Avg	*70.88*	*57.07*	*60.42*	
GPT-4o	Safe-to-continue	80.29	34.92	48.67	$0.17
	High-Risk	24.91	71.58	36.96	
	Weighted Avg	*67.46*	*43.41*	*45.96*	
Ours	Safe-to-continue	**83.09**	**71.75**	**77.00**	**$0.03**
	High-Risk	**35.51**	51.58	42.06	
	Weighted Avg	***72.06***	***67.07***	***68.91***	

(c) Our test set (EnterpriseBench)

Model	Class	Precision	Recall	F1-Score	Fore. Cost
Claude-4.5 Sonnet	Safe-to-continue	0.96	0.52	0.67	$0.74
	High-Risk	0.14	0.81	0.24	
	Weighted Avg	*0.89*	*0.54*	*0.64*	
Gemini-2.5 Pro	Safe-to-continue	0.96	0.51	0.66	$0.59
	High-Risk	0.14	0.78	**0.33**	
	Weighted Avg	*0.88*	*0.53*	*0.63*	
Qwen-3 8B	Safe-to-continue	0.98	0.17	0.29	**$0.01**
	High-Risk	0.10	0.96	0.19	
	Weighted Avg	*0.89*	*0.24*	*0.28*	
GPT-4o	Safe-to-continue	0.98	0.50	0.66	$0.59
	High-Risk	0.16	0.89	0.27	
	Weighted Avg	*0.89*	*0.54*	*0.62*	
Ours	Safe-to-continue	**0.98**	**0.57**	**0.72**	**$0.01**
	High-Risk	0.17	0.80	0.29	
	Weighted Avg	***0.91***	***0.60***	***0.68***	

(d) MBPP

Table 5. **Task failure forecasting performance across four evaluation settings.** (a) Who and When (GAIA + AssistantBench), then we have held-out synthetic test set on (b) GAIA + HotpotQA, (c) SWE-Bench, and (d) EnterpriseBench. Class "High-Risk" corresponds to steps that leads to failure if agent continues under current policy ($y = 1$), while "Safe-to-continue" corresponds to safe or recoverable steps ($y = 0$). "Fore. Cost" denotes total inference cost for forecasting. Blue rows highlight our method.

categories, validating the quality of the synthetic supervision.

> ***RQ5:*** *Can recoverable and high-risk errors be reliably differentiated solely through partial trace inspection?*

To characterize the intrinsic difficulty of failure forecasting, we investigate whether the "fatality" of an error is evident from partial trace inspection alone, or requires modeling future agent-environment dynamics.

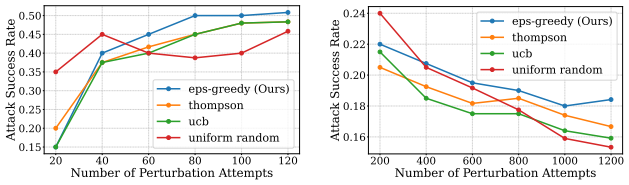
Human Study. We conducted an annotation study where two annotators each evaluated 100 partial traces from our test set (36 No Error, 36 Recoverable, 28 High-Risk), predicting whether each would result in task success, or failure. Table 11 shows that while annotators reliably detect error-free actions (82.9% accuracy), they struggle to distinguish recoverable from high-risk errors (53.9% accuracy, near chance). This suggests that error severity is not immediately apparent from local trace context alone.

Embedding Analysis. To complement the human study, we analyze the latent geometric structure of errors by visualizing t-SNE embeddings of the final agent action across all training traces (Figure 2). While "No Error" actions form coherent clusters, "Recoverable" and "High-Risk" errors are deeply intermingled. This topological overlap further shows that recoverability is not a surface-level property of action itself, but rather a latent property of how environment state evolves and whether policy π can recover from that state.

Takeaway: *Error severity is not linguistically or semantically distinct from action content alone. Thus, accurate forecasting requires modeling the agent-environment inter-*

Benchmark	Class	Support	Ours	Baseline	Δ	95% CI
Our test set (GAIA + HotpotQA)	High-Risk	50	28.57	16.29	+12.28	[+1.02, +25.50]
	Safe-to-continue	430	92.92	40.16	+52.76	[+47.20, +58.50]
Who and When (Zhang et al., 2025c)	High-Risk	58	16.23	12.01	+4.22	[+0.08, +8.51]
	Safe-to-continue	879	82.28	8.55	+73.73	[+70.56, +76.75]
Our test set (MBPP)	High-Risk	27	26.88	18.71	+8.18	[+5.38, +11.18]
	Safe-to-continue	270	66.67	28.48	+38.19	[+31.66, +44.98]
Our test set (EnterpriseBench)	High-Risk	95	42.06	40.54	+1.52	[-0.59, +5.99]
	Safe-to-continue	315	77.00	66.41	+10.59	[+7.20, +14.01]

Table 6. F1 comparison between Finetuned-Forecaster and Qwen3-8B across benchmarks. Deltas are reported as Ours − Baseline, with 95% confidence intervals from 1000 bootstrap samples.



(a) GAIA Dataset *(b)* HotpotQA Dataset

Table 7. **Cumulative ASR of perturbation-selection strategies.** On GAIA, ASR increases over time with ε-greedy achieving highest ASR. On HotpotQA, all methods show decreasing ASR, yet ε-greedy consistently maintains highest.

HotpotQA Dataset				
Category	**Failure Category**	**Strategy**	**ASR (%)**	**Attacks**
Top-3 by Baseline	Disobey task spec.	Random (Baseline)	17.7	135
		Bandit (Ours)	18.0	**325**
	Disobey role spec.	Random (Baseline)	17.5	130
		Bandit (Ours)	17.3	**200**
	Reasoning action mismatch	Random (Baseline)	17.0	110
		Bandit (Ours)	20.0	**780**
Bottom-3 by Baseline	Premature termination	Random (Baseline)	13.0	130
		Bandit (Ours)	7.7	**15**
	Incorrect verification	Random (Baseline)	11.0	125
		Bandit (Ours)	10.0	**25**
	Unaware of term. cond.	Random (Baseline)	8.3	120
		Bandit (Ours)	0.0	**5**

GAIA Dataset				
Category	**Failure Category**	**Strategy**	**ASR (%)**	**Attacks**
Top-3 by Baseline	Disobey task spec.	Random (Baseline)	57.1	30
		Bandit (Ours)	51.4	**145**
	Loss of conv. history	Random (Baseline)	51.2	31
		Bandit (Ours)	40.0	**40**
	Premature termination	Random (Baseline)	45.2	32
		Bandit (Ours)	50.0	**20**
Bottom-3 by Baseline	Unaware of term. cond.	Random (Baseline)	35.7	28
		Bandit (Ours)	25.0	**5**
	Incorrect verification	Random (Baseline)	35.5	32
		Bandit (Ours)	5.0	**5**
	Step repetition	Random (Baseline)	31.2	30
		Bandit (Ours)	40.1	**22**

Table 9. **Bandit Attack Allocation Analysis.** Performance on top-3 and bottom-3 failure categories, ranked by Random baseline ASR. Darker cells indicate more attacks.

action trajectory.

> ***RQ6:*** *Can the forecasting model generalize to different agent policies?*

Our forecasting model is trained on traces generated by a specific agent policy π (Magentic-One with GPT-4o). A natural question is whether the learned failure patterns transfer when the underlying agent behavior changes. To investigate this, we evaluate our forecaster's effectiveness when deployed with an out-of-domain policy: Magentic-One using GPT-4o-mini instead of GPT-4o as the backbone LLM.

Table 12 shows that our forecaster maintains utility even under policy shift. On the in-domain policy (GPT-4o), our forecaster improves accuracy from 70% to 74% via selective inference scaling. When deployed with the out-of-domain policy (GPT-4o-mini), the forecaster still provides meaningful gains, improving accuracy from 47% to 53%. While the absolute performance is lower (due to the weaker base policy), the relative improvement demonstrates that the learned failure patterns partially transfer across LLM backbones.

However, these results should be interpreted cautiously. Our current training procedure generates data using a single fixed policy (Magentic-One with GPT-4o), which means the forecaster learns to predict failures specific to that policy's behavior patterns. The partial transfer to GPT-4o-mini suggests some failure modes are policy-invariant (e.g., constraint violations, information loss), but others may be policy-specific (e.g., reasoning errors that depend on model capabilities).

Takeaway: *Our forecaster shows promising generalization to policy variations (different LLM backbones within the same MAS architecture), but was trained on a single fixed policy. A comprehensive study of cross-policy generalization and extensions to train policy-agnostic forecasters from multi-policy training data remain important directions for future work.*

Dimension	Metric / Scale	Score
Study 1: Plausibility & Realism (Mixed Samples)		
Trace Coherence	Mean Score (1–3)	2.45 $\pm$0.07%
Action Plausibility	Mean Score (1–3)	2.45 $\pm$0.07%
Study 2: Fault Injection Quality (Perturbed Only)		
Category Alignment	Mean Score (1–3)	2.49 $\pm$0.07%

Table 10. **Human Evaluation of Synthetic Trace Quality.** We report the Mean Scores on a 3-point scale (1=Not, 2=Partial, 3=Complete). *Action Plausibility* measures whether the final step resembles a realistic LLM output. *Category Alignment* measures whether the generated error semantically matches the requested failure category.

No-Error Acc.	Severity Acc.
82.9% $\pm$6.9%	53.9% $\pm$9.8%

Table 11. **Human Annotation Performance.** We report the average accuracy across annotators. *Severity Acc.* measures ability to distinguish between recoverable & high-risk errors, where 50% indicates random guessing 100% indicates perfect differentiation. *No-Error Acc.* - accuracy w.r.t. 'No error' samples.

RQ7: *Under the same compute budget (number of rollouts), does it help prioritizing diversity over label precision as argued in section 8?*

In section 8, we theoretically argued that under fixed computational budgets, maximizing state coverage with single rollouts ($N = 1$) yields better generalization than performing multiple rollouts per state ($N > 1$). Here, we empirically validate this claim by training forecasters under identical total compute budgets but different allocation strategies.

We compare three training configurations: (1) our default approach using single rollouts per perturbed state ($N = 1$), achieving maximum state diversity; (2) an alternative using three rollouts per state ($N = 3$) for higher label precision but covering only one-third as many unique states; and (3) the base Qwen-3-8B model for reference. Critically, configurations (1) and (2) consume the same total number of rollouts—we simply reallocate the budget between breadth (more states) versus depth (more rollouts per state).

Table 13 shows that the diversity-prioritizing strategy ($N = 1$) outperforms the precision-focused approach ($N = 3$): 78.01% vs. 52.63% weighted F1 on the human-annotated "Who and When" benchmark. The $N = 1$ model achieves dramatically higher performance on the Safe-to-continue class (82.28% vs. 55.43% F1), demonstrating superior generalization to safe states. While both trained models struggle with the rare High-Risk error class relative to their performance on common cases, the diversity-focused model shows a better overall balance.

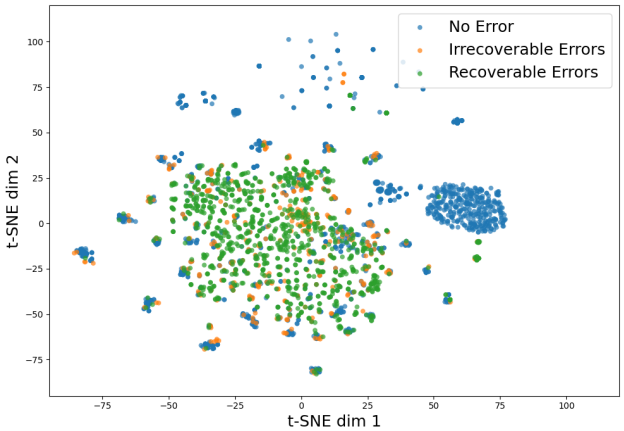


Figure 2. **t-SNE Visualization of Last Agent Action Embeddings.** We visualize the representations of the last action across traces from our generated training set, color-coded by their ground-truth error status. While benign actions ("No error") cluster, recoverable and high-risk errors overlap significantly. This shows detecting irrecoverability requires analyzing the agent trace's causal future, not just the action's content.

Agent Policy π		Forecaster/Strategy	Accuracy
MAS	LLM		
Magentic-One	GPT-4o	None (No Scaling)	0.70
		Ours (FT 8B) + BoN	**0.74**
Magentic-One	GPT-4o -mini	None (No Scaling)	0.47
		Ours (FT 8B) + BoN	**0.53**

Table 12. **Impact of Agent Policy on Forecasting Model.** We study the performance of our forecasting model across different base agent policies (π) by varying the LLM backbone, GPT-4o as the in-domain policy and GPT-4o-mini as the out-of-domain policy, within the Magentic-One multi-agent system architecture. We evaluate the downstream application of inference scaling on the HotpotQA held-out set of "hard" complexity queries.

Takeaway: *Empirical results validate our theoretical analysis from section 8: under fixed computational budgets, prioritizing state coverage with single rollouts outperforms averaging multiple rollouts per state. The diversity-focused approach achieves 78.01% weighted F1 compared to 52.63% for the precision-focused alternative, validating that generalization error from limited state coverage dominates label noise in high-dimensional agent state spaces.*

12. Conclusion, Limitations and Future Work

Distinguishing recoverable from fatal errors is critical for reliable LLM agents, yet existing methods largely diagnose failures post-hoc. We introduce *Task Failure Forecasting* with adaptive fault injection to train lightweight forecasters that predict high-risk failures online from partial traces. Our forecaster outperforms general-purpose LLM monitors at substantially lower cost and enables cost-aware inference-

Model	Class	F1 Score
Ours (# rollouts = 1)	Safe-to-continue	**82.28**
	High-Risk	**16.23**
	Weighted Avg	***78.01***
Ours (# rollouts = 3)	Safe-to-continue	55.43
	High-Risk	12.44
	Weighted Avg	*52.63*
Qwen3-8B	Safe-to-continue	8.55
	High-Risk	12.01
	Weighted Avg	*8.78*

Table 13. **Empirical Validation of Diversity vs. Precision Trade-off.** We compare forecasting performance under identical computational budgets but different allocation strategies. Our default approach, $N = 1$ rollout per state, maximizes state coverage, while the alternative, $N = 3$ rollouts per state, prioritizes label precision at the cost of seeing fewer unique states. Both consume the same total number of rollouts. Evaluation on the "Who and When" human-annotated benchmark confirms that state diversity yields better generalization than label precision in high-dimensional agent state spaces.

time intervention across web, coding, and agentic tasks. A key limitation is policy dependence: recoverability is defined relative to the corrective capabilities of the agent policy used during data generation. We therefore focus on the fixed-policy setting, the most direct deployment scenario for runtime monitoring and intervention, while extending to policy-conditional or foundation forecasters that generalize across policies remains important future work.

13. Prompts

This section provides the complete prompt templates used throughout our data generation pipeline (subsection 3.1) and evaluation procedures (section 5). These prompts instantiate the LLM-based components of our framework: failure analysis ($\mathcal{M}_{\text{sel}}$ for vulnerability selection), attack generation (perturbation operator Π), LLM-as-a-judge (for measuring task success) and Failure forecasting.

Agent Failure Analysis Prompt

```
You are an expert AI agent evaluator.
Your task is to analyze the following
FAILED execution trace and identify
the root cause of the failure w.r.t.
the given "Failure Categories".

(One small caveat - the agent only
has access to bing search, so it can
only use that and not any other search
engines)

**Failure Categories:**

- Disobey task specification: Failure
to adhere to the specified constraints
or requirements of a given task,
leading to suboptimal or incorrect
outcomes.

- Disobey role specification:
Failure to adhere to the defined
responsibilities and constraints of
an assigned role, potentially leading
to an agent behaving like another.

- Step repetition: Unnecessary
reiteration of previously completed
steps in a process, potentially
causing delays or errors in task
completion.

- Loss of conversation history:
Unexpected context truncation,
disregarding recent interaction
history and reverting to an antecedent
conversational state.

- Unaware of termination conditions:
Lack of recognition or understanding
of the criteria that should trigger
the termination of the agents
interaction, potentially leading to
unnecessary continuation.

- Conversation reset: Unexpected
or unwarranted restarting of a
dialogue, potentially losing context
and progress made in the interaction.

- Fail to ask for clarification:
Inability to request additional
information when faced with unclear or
incomplete data, potentially resulting
in incorrect actions.

- Task derailment: Deviation from
the intended objective or focus of a
given task, potentially resulting in
irrelevant or unproductive actions.

- Information withholding: Failure
to share or communicate important data
or insights that an agent possess and
could impact decision-making of other
agents if shared.

- Ignored other agents input:
Disregarding or failing to adequately
consider input or recommendations
provided by other agents in the
system, potentially leading to
suboptimal decisions or missed
opportunities for collaboration.
```

```
- Reasoning-action mismatch:
Discrepancy between the logical
reasoning process and the actual
actions taken by the agent,
potentially resulting in unexpected
or undesired behaviors.

- Premature termination: Ending a
dialogue, interaction or task before
all necessary information has been
exchanged or objectives have been met,
potentially resulting in incomplete or
incorrect outcomes.

- No or incomplete verification:
(partial) omission of proper checking
or confirmation of task outcomes or
system outputs, potentially allowing
errors or inconsistencies to propagate
undetected.

- Incorrect verification: Failure to
adequately validate or cross-check
crucial information or decisions
during the iterations, potentially
leading to errors or vulnerabilities
in the system.
---

**Failed Task Execution History:**
---
{history_str}
---

**Correct Answer to the User Query**
---
{gold_ans}
---

Please identify the specific steps
(multiple steps are allowed) where the
agent made a critical error that led
to the final failure.

Return your analysis in the following
JSON format ONLY. Your entire response
must be a single, valid JSON object,
starting with {{ and ending with
}}. Do not include any other text,
explanations, or markdown formatting.

{{
"errors": [
{{
"step_index": <integer index of the
step where the error occurred>,
"agent_name": "<the agent that made
the mistake at this step>",
"reasoning": "<An explanation of
why the agent action was faulty and
reasoning for categorization of the
error>"
"error_category": "<the most relevant
failure category from the list>",
"evidence": "<Give verbatim text
from the trace to properly show the
evidence for the mistake>"
"context_before_failure": "<to
understand the faulty action, give the
subtask that was given to the faulty
subagent here. Add more concrete
context as you feel is necessary>"
"faulty_action": "<the exact content
of the action that was incorrect -
verbatim copy the information.>",
}}
]
}}
```

Critical Steps Identification Prompt

```
You are an expert in AI agent
evaluation. Analyze the following
successful execution trace and
identify the {k} most critical steps.

A "critical" step is one where a
small, plausible mistake could cascade
into a complete task failure.

Exclude step 0 (user query) and avoid
the final step if possible.

**Execution History:**
---
{history_str}
---

Return your answer in JSON format:

{{"critical_steps": [{{"step_index":
<int>, "reason": "<text>"}}]}}
```

Attack Generation Prompt

```
You are an expert red-teamer for AI
agents. Your goal is to devise a
subtle, yet natural, modification for
a GIVEN step in a successful execution
that will realistically cause it to
fail.

**Task Information:**
- Task: {task_info['query']}
- Ground Truth:
{task_info['groundtruth_answer']}

**Targeted Failure Category:
{category}**
- Description: {CATEGORY_DESCRIPTIONS.get(category,
"N/A")}

Here are some examples of real
failures in this category:
```

```
---
{example_str}
---

**Original Execution History:**
---
{history_str}
---

Your task is to generate a new,
faulty action for **step number
{attack_step}**, inspired by the
examples.

The attack MUST be a plausible mistake
for an LLM agent to make, and it
should stay within the realm of the
responsibilities/role of the agent
being attacked.

Provide your response in this exact
JSON format:

{{
"attack_step": {attack_step},
"failure_category": "{category}",
"attack_content": "<SPECIFIC and
complete modified action for the agent
at this step>",
"reason": "<Detailed explanation
of why this modified action is a
plausible error and how it leads to
failure, referencing the category>"
}}
```

LLM-as-a-judge Prompt

```
For the query: {task}

Ground truth: {ground_truth}

Model's prediction:
{predicted_answer}

**Rubric for Comparison:** Compare the
model's predicted answer to the ground
truth. Consider factual accuracy, not
just surface text match.

Provide a concise explanation of
why the prediction is correct or
incorrect.

Give a score of 1 if the prediction
matches the ground truth, otherwise 0.

The output must be in the format:

Thought: "<concise justification>",
Score: <1 or 0>
```

Failure Forecasting Prompt

```
You are an expert analyzing
multi-agent system (MAS) executions.
I will provide the an task/question
and a history of the MAS execution.
Each step in the history has a "step"
number and describes what the agent(s)
did.

Your task is to carefully analyze the
*last step* in the provided execution
history. Based on this last step, the
overall Original Task/Question, and
the execution history (till current
step, need not be the full execution
history), determine if continuing the
execution from this point is likely
to lead to a *task failure*. Your
assessment should focus on forecasting
potential failure, considering if the
state achieved by the last step or the
action taken in the last step, under
the context of the previous steps,
steers the MAS toward an incorrect
path or makes successful completion
improbable.

Output the answer in exactly this
format (no extra text):

Direction : <Correct or Incorrect>
Reason : <Reason behind your
prediction of above direction>

Here is the information:

Original Task/Question:
{query}

Execution History:
{formatted_history}
```

14. Human Evaluation Details

This section provides additional details on the human evaluation studies reported in RQ4 and RQ5. Figure 3 and Figure 4 show the annotation interface used by annotators to assess trace plausibility (Study 1) and fault injection quality (Study 2), supporting the quantitative results presented in Table 10 and Table 11.

For conducting the human evaluation, we recruited two annotators with prior experience in evaluating LLM agent traces to conduct the evaluation using a custom web interface. For Study 1 (Plausibility & Forecasting), the annotators evaluated a set of 100 randomly sampled traces from the test set in a "blind" mode (Figure 3), where they were unaware whether a trace was original or synthetically perturbed. They were instructed to review the agent exe-

cution log against the user request and evaluate the final step using a 3-point Likert scale on two dimensions: *Trace Coherence* (1=Not Coherent, 2=Partially Coherent, 3=Completely Coherent) and *Action Plausibility* (1=Implausible, 2=Partially Plausible, 3=Completely Plausible). Additionally, they provided a binary judgment (*Error Detection*) on whether the step constituted an error leading to task failure. For Study 2 (Fault Injection Quality), the annotators evaluated 100 perturbed traces (Figure 4). In this setting, they were explicitly informed that the final step was a synthetic perturbation intended to reflect a specific failure category (e.g., "Disobey task specification"). They were provided with the specific error definition and instructed to rate the *Alignment Score* on a 3-point scale (1=Misaligned, 2=Partially Aligned, 3=Completely Aligned) based on how well the generated perturbation linguistically and functionally matched the intended error type.

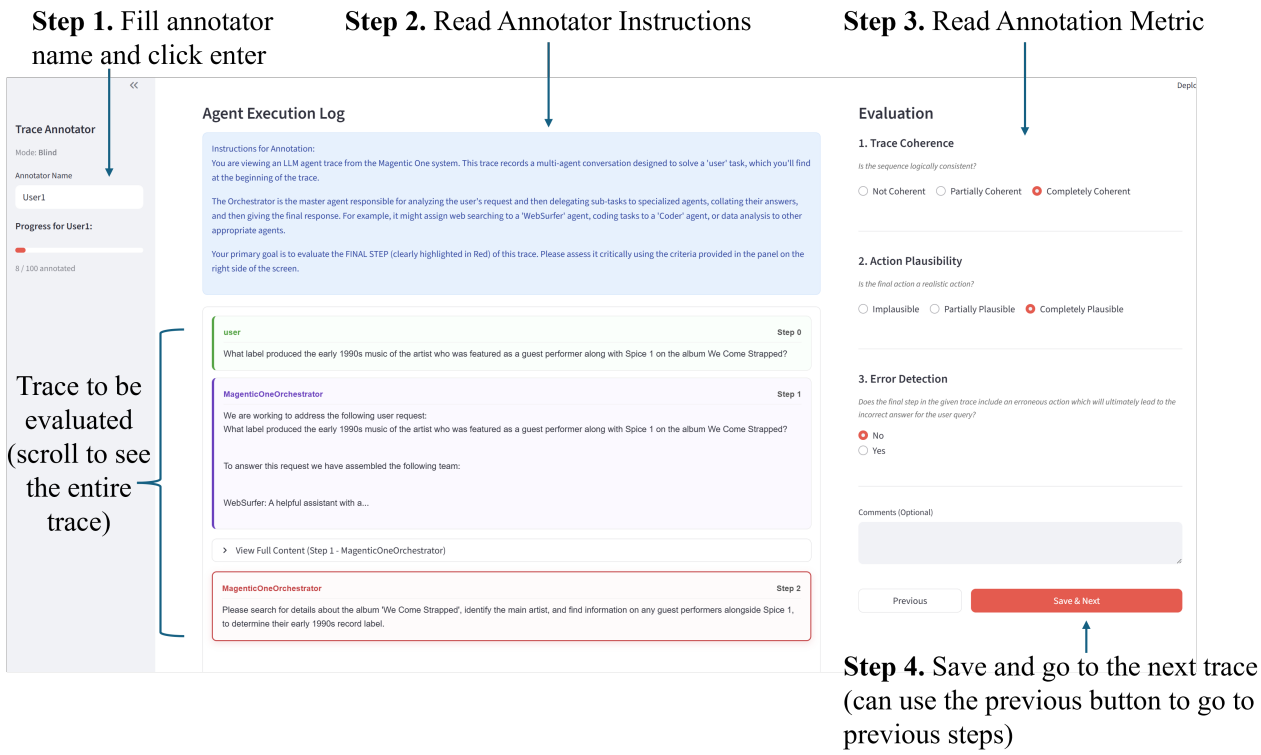


Figure 3. **Data Annotation Application View for Study 1**

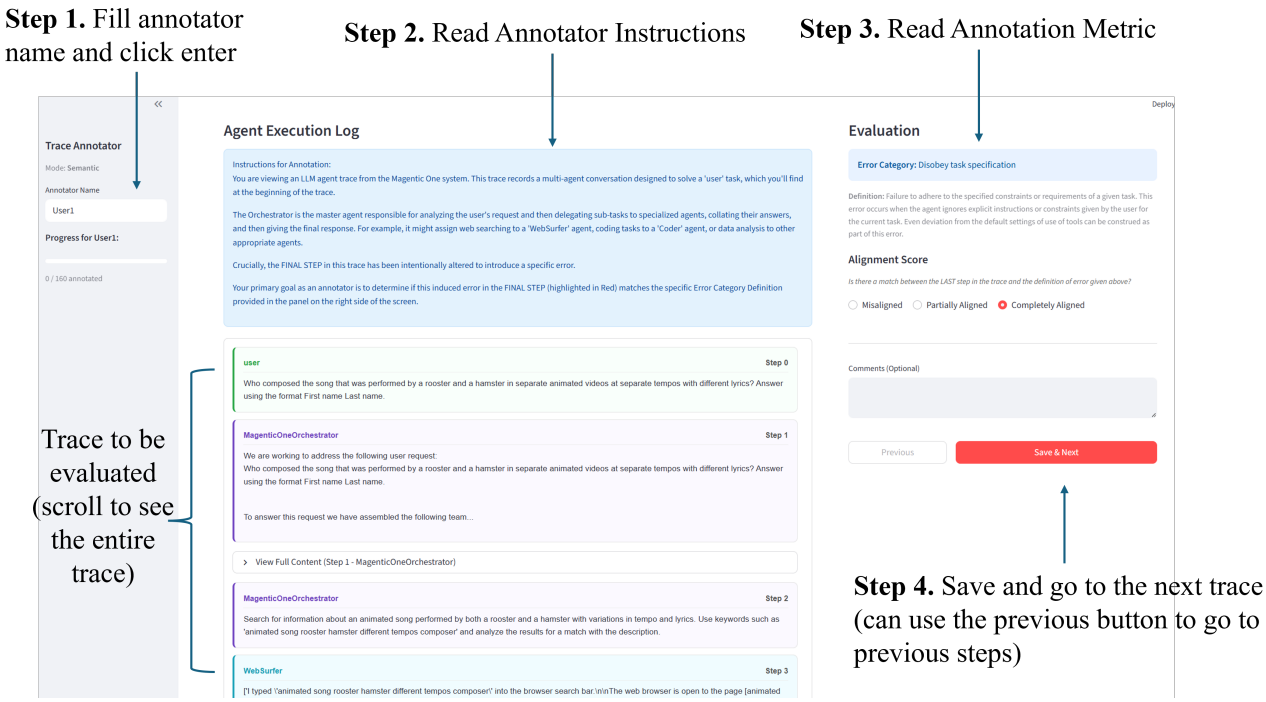


Figure 4. **Data Annotation Application View for Study 2**