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]

[1]Fujitsu Research [2]Carnegie Mellon University

[*]Equal contribution

Abstract

LLM agents often execute long-horizon tasks where sequential tool calls and reasoning steps compound into a final outcome. When an agent errs mid-execution, the critical question is not whether an error occurred, but whether recovery under the current policy is still possible. Existing failure analysis answers this retrospectively, making it too late for runtime intervention. We introduce **Task Failure Forecasting**: predicting from a partial execution trace whether continued execution under the current agent policy is likely to fail. This is hard: human experts achieve only 54% accuracy at distinguishing recoverable from failure-inducing errors, and frontier models such as Claude-4.5 Sonnet reach 59.87% weighted F1. The core obstacle is that failed traces mix recoverable mistakes with fatal ones, obscuring which steps drive downstream failure. We address this with **adaptive fault injection**: injects targeted perturbations into successful traces, uses bandit-based prioritization of high failure yield error types and rollout-based verification providing policy-conditioned step-level supervision. A lightweight Qwen-3-8B model trained on this synthetic data achieves 78.01% weighted F1 on human-annotated benchmarks, outperforming Claude-4.5 Sonnet by 18.1 weighted-F1 points at 100× lower forecasting cost. When deployed as a runtime monitor for selective inference-time intervention, the forecaster improves agent success rates across HotpotQA, MuSiQue, GAIA, AssistantBench, SWE-Bench, MBPP and EnterpriseBench, yielding gains in the range of 2–16 % points across held-out and unseen tasks.

1 Introduction

Recent LLM agents operate over long horizons [17, 10], use external tools [32, 6], and interact with evolving environments. As tasks unfold across many sequential decisions, the consequences of a mistake are not uniform: some mistakes can be corrected by later agent actions, while others make successful completion unlikely under current policy. Distinguishing these cases during execution is important because it determines whether the agent should continue normally or allocate additional intervention compute, such as reflection [22], resampling [29] or human handoff.

However, the dominant paradigm for analyzing agent failures remains fundamentally *post-hoc*. Human annotators typically inspect completed traces and identify root causes retrospectively [36], while automated LLM-as-judge systems and recent fault-localization methods [34, 11] likewise operate after execution has already finished. These methods are useful for attribution, debugging, and

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

evaluation, but they do not answer the question that matters at inference time: *given the current partial trace of an agent, is continued execution under present policy likely to succeed?* Strikingly, this question is difficult even for humans: in our annotation study, expert annotators asked to distinguish recoverable from high-risk errors on partial traces achieve only 54% accuracy - barely above chance - underscoring that error severity is not apparent from local trace context alone.

Task Failure Forecasting differs from prior failure analysis in both timing and target. Fault-localization methods identify which step caused an already completed failure; process-supervision methods [37] judge local step quality. In contrast, our goal is to predict before termination whether the current partial trace remains recoverable under the deployed policy. 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.

We formalize 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, the goal is to estimate future failure probability $f_\theta(\tau_{\leq t}) := P(Y(\tau) = 1 \mid \tau_{\leq t}, \pi)$, where $Y(\tau)$ denotes terminal task outcome and $Y(\tau) = 1$ implies task failure. Failure risk is therefore policy-conditioned rather than absolute: the same prefix may be recoverable for a stronger policy but high-risk for a weaker one. Throughout the paper, "high-risk" refers to prefixes whose continuation is likely to fail under policy being monitored.

Training such a forecaster from naturally occurring traces is non-trivial. A natural approach is Monte Carlo rollout [15, 27]: from each trace prefix, continue execution multiple times, estimate success probability from the observed outcomes, and train a model to predict when that probability is near zero. In practice, however, this is computationally prohibitive and expensive for realistic agent traces, which are long [10, 36]. At the same time, learning directly from naturally failed trajectories creates a credit-assignment problem, since failed traces often contain a mixture of benign mistakes and genuinely task-dooming ones, making it unclear which step actually caused the loss of recoverability.

We address this challenge with *adaptive fault injection*. Rather than disentangling mixed signals in naturally failed traces, we start from successful traces and inject targeted perturbations at selected intermediate steps. A multi-armed bandit adaptively prioritizes error categories that are more likely to yield high-risk prefixes, while rollout-based verification labels whether continuation from a perturbed prefix leads to downstream failure under the policy. This gives step-localized stochastic supervision for learning the failure-risk boundary. 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 not generated by our pipeline. The resulting synthetic data supports training a lightweight forecaster that can run alongside agent as an inexpensive runtime monitor.

Empirically, we find that online forecasting is difficult. Beyond humans performing near chance, even frontier LLMs used as forecasters remain limited: Claude-4.5 Sonnet reaches 59.87% weighted F1 at $17.32 per test set, and GPT-4o reaches 68.66% at $9.94. A lightweight Qwen-3-8B model fine-tuned solely on our synthetic data achieves 78.01% weighted F1 on the same human-annotated benchmark [36], outperforming all frontier models at $0.13, approximately 100× lower inference cost. When deployed as a runtime monitor for selective inference-time intervention, the forecaster also improves downstream agent success rates on both held-out and previously unseen tasks (HotpotQA [31]: 70% → 74%; MuSiQue [24]: 12% → 24%; Who&When (GAIA, AssistantBench) [36]: 10% → 14%; SWE-Bench [10]: 67% → 69%; MBPP [1]: 47% → 49%; EnterpriseBench [26]: 66% → 82%;).

We summarize our main contributions as follows:

- We formalize *Task Failure Forecasting* as online prediction of policy-conditioned high-risk failure states from partial agent traces, distinct from post-hoc fault attribution and local process supervision.
- We propose *adaptive fault injection*, combining bandit-based perturbation selection with rollout-based stochastic verification, as a principled method to generate step-level supervision that targets future recoverability rather than local action correctness. We also provide a theoretical analysis of the compute tradeoff between broad state (trace-prefix) coverage and precise estimation via repeated rollouts on fewer states under fixed budgets.
- We show that a lightweight Qwen-3-8B forecaster trained on this synthetic supervision outperforms frontier LLMs on human-annotated benchmarks at 100× lower cost, and enables effective runtime intervention that improves agent success rates across webQA tasks (HotpotQA, MuSiQue, GAIA, AssistantBench), coding tasks (SWE-Bench, MBPP) and enterprise agent tasks (EnterpriseBench).

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 an 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 the probability that the 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 a reward sequence such that $r_k = 0$ for all $k < T$ and $r_T = Y(\tau)$. Then the return from any partial prefix is exactly the final failure indicator. Accordingly, the forecaster can be interpreted as approximating a 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 the 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.

3 Methodology

To train a failure forecaster from partial traces, we need labeled examples that map a trace prefix to its eventual task outcome. However, naturally occurring failed traces do not provide this supervision cleanly: they may contain multiple deviations before termination, and some of them may still be recoverable under the policy while others may destroy the remaining probability of success. Therefore, the terminal task outcome alone does not reveal which step caused the loss of recoverability.

To clarify this distinction, we first formalize what we mean by recoverable versus high-risk errors. Let $\mathcal{R}(\tau_{\leq t}; \pi)$ denote a stochastic rollout function that completes a partial trace $\tau_{\leq t}$ using agent policy π. The rollout is stochastic because the policy π is stochastic: repeated continuations from the same prefix can produce different actions and hence different final outcomes. An error is **recoverable** if the policy π has the capacity to succeed: $P(Y(\mathcal{R}(\tau_{\leq t}; \pi)) = 0) > 0$. Conversely, an error is **irrecoverable** at step t if the success probability under π becomes zero: $P(Y(\mathcal{R}(\tau_{\leq t}; \pi)) = 0) = 0$. In practice, exact irrecoverability cannot be established from finitely many stochastic rollouts. We therefore use **high-risk** to refer to trace prefixes whose success probability under the current policy is close to zero. Finite rollout supervision should therefore be interpreted as noisy evidence of high-risk failure states rather than proof of exact irrecoverability. For instance, repeating a command (e.g., `$ ls -la`) is recoverable, since the agent can still proceed to complete the task. However, prematurely terminating before satisfying the user's request is irrecoverable, as no future action under the current policy can correct the outcome; operationally, such cases are treated as high-risk when rollout evidence indicates that successful recovery is very unlikely.

We hypothesize that effective failure forecasting requires disentangling recoverable mistakes from high-risk errors to focus learning on the root causes of irrecoverability. To this end, we propose an *adaptive fault injection* framework that generates supervised training data by systematically introducing targeted perturbations into successful agent traces. Concretely, our approach identifies vulnerable steps in successful traces and employs an *adaptive attacker* to inject *plausible errors*, then evaluates the resulting outcome via environment rollouts (Figure 1). This yields step-level stochastic labels that sharpen the link between a specific action and downstream task failure, enabling the forecaster to learn the boundary of task irrecoverability. Section 3.1 details this data generation scheme and Section 3.2 describes how we leverage this supervision to train a lightweight model.

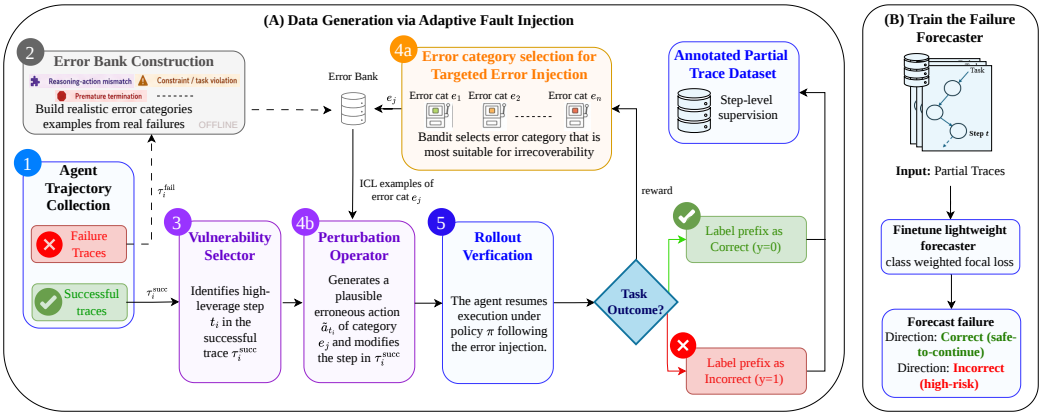


Figure 1: **Adaptive fault injection for partial-trace supervision.** (1) Successful trajectories provide base traces, while failed trajectories are used offline for error-bank construction. (2) The error bank stores realistic failure categories and examples. (3) A vulnerability selector identifies high-leverage steps in a successful trace. (4a) A bandit selects an error category likely to induce high-risk failures. (4b) A perturbation operator injects a plausible faulty action (using error bank). (5) The agent rolls out from 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.

3.1 Data Generation via Adaptive Fault Injection

To operationalize the distinction between recoverable and high-risk errors, we synthesize a dataset with stochastic "root cause" labels by injecting plausible faults into successful traces. This allows us to observe the downstream effect of a single perturbation type in isolation, evaluate the resulting continuation via Monte Carlo rollouts, and obtain dense, step-level supervision for failure forecasting.

(i) Error Bank Construction For Plausible Faults. To ensure synthetic errors reflect realistic agent failures rather than arbitrary noise, we construct an *Error Bank* $\mathcal{B}$ based on failure analysis of common LLM agent error categories [4], e.g., "Reasoning-Action Mismatch", "Constraint Violation". Concretely, given these categories and a failed trace, an LLM is prompted to infer instances of each error type, producing representative examples for fault injection.

(ii) Fault Injection. To efficiently navigate the vast space of potential perturbations, we first employ a *Vulnerability Selector* $\mathcal{M}_{\text{sel}}$, implemented with an LLM to identify high-leverage steps in a successful trace that are susceptible to critical failure (e.g., key reasoning steps or API calls). For a selected target step t, we use a Multi-Armed Bandit (MAB) policy, π_{bandit}, to select an error category c from the error categories in the error bank $\mathcal{B}$, prioritizing categories that more often induce failure. The bandit follows an ϵ-greedy policy (analyzed in RQ3 in Section 6), receiving reward $r = 1$ if a rollout yields task failure and $r = 0$ otherwise. Since rollouts are computationally expensive and error categories can have different failure rates that are unknown a priori, the bandit adaptively concentrates sampling on high-yield error types rather than uniform exploration, enabling efficient dataset construction with limited computational budget. A perturbation operator LLM $\Pi(\cdot)$ then synthesizes a plausible erroneous action $\tilde{a}_t$ conditioned on the selected error category c and the pre-action history h_t.
This active selection concentrates compute on generating valid *high-risk* examples rather than benign mistakes. In other words, to efficiently train the forecasting model f_θ to identify transition boundaries (as stated in Section 2), we wish to minimize loss over the distribution of high-risk errors, $\mathcal{P}_{\text{hard}}(s_t, y)$, rather than the uniform distribution of states where failures are sparse (the true data generating process $\mathcal{P}_{\text{true}}(s_t, y)$). The bandit policy π_{bandit} acts as an adaptive sampler that approximates this target distribution by concentrating probability mass on error types with high failure rates. We treat the resulting dataset $\mathcal{D}$ as a fixed dataset and analyze learning performance with respect to the underlying distribution induced by the bandit sampler, $\mathcal{P}_{\text{hard}}$. Note that $\mathcal{P}_{\text{hard}}$ is not intended to match the natural frequency of agent errors; it deliberately over-samples error categories that yield informative high-risk states under a fixed rollout budget, improving class balance and state coverage. We show that this yields better forecasters than uniform category sampling (RQ3, Table 4) and that the resulting forecaster transfers to naturally occurring failures outside the Error Bank (Appendix 14).

(iii) Rollout-Based Stochastic Labeling. Determining the exact probability of task failure $P(Y(\mathcal{R}(\tau_{\leq t}; \pi)) = 1)$ requires infinitely many rollouts. However, for the purpose of training

a parameterized function approximator f_θ, we do not require precise probability estimates for every individual sample. Instead, we treat the outcome of a *single* Monte Carlo rollout as a stochastic label sampled from the underlying success probability distribution. Let $y_{\text{roll}} \in \{0, 1\}$ denote the rollout failure indicator, where $y_{\text{roll}} = 1$ means the rollout ended with task failure and $y_{\text{roll}} = 0$ means otherwise. Then $\mathbb{E}[y_{\text{roll}} \mid \tau_{\leq t}] = P(Y(\tau) = 1 \mid \tau_{\leq t}, \pi)$.
Thus, a single rollout provides a noisy but unbiased sample of the policy-conditioned failure outcome. Training on these binary realizations allows f_θ to learn the expected failure probability surface. We use the same capable policy π (that uses a powerful LLM) that generated the original successful traces for rollouts, ensuring that the resulting labels reflect the behavior of the policy we ultimately wish to monitor. As we demonstrate in Section 4, this single-rollout approach can still be effective provided dataset covers a sufficiently diverse support of state space. Full procedure in Algorithm 1.

3.2 Training

We train a lightweight binary forecaster f_θ to predict whether a partial execution trace has entered a high-risk state under policy π. Given a partial trace $\tau_{\leq t}$, the model outputs one of two labels: *Safe-to-continue* $(y = 0)$, which includes both correct/no-error prefixes and recoverable-error prefixes, or *High-Risk* $(y = 1)$, indicating that continuation under π could lead to task failure. We fine-tune a Qwen3-8B [30] in this work to act as the forecaster f_θ. The forecaster uses the standard causal language-modeling head of Qwen3-8B; no separate classifier head is added. Given the task and the serialized partial trace, it generates a structured response `Direction: <Correct|Incorrect>` followed by `Reason: <...>` (prompt in Section 15). The `Direction` field is the binary forecast (`Correct`$\rightarrow y{=}0$, `Incorrect`$\rightarrow y{=}1$); the generated reason is not used in computing training loss.

Partial Trace Dataset Construction. For each failed trace τ with a failure-inducing step at t_{cause} , we define training examples as: $\{\tau_{\leq t}, label = 0\}$ for all sub-traces before the root cause $(t < t_{\text{cause}})$, and $\{\tau_{\leq t_{\text{cause}}}, label = 1\}$ for the critical transition point. This yields a supervised dataset $\mathcal{D}$ where each example maps a partial trace to its recoverability label.
In summary, the resulting dataset contains three semantic sample types: *Correct/no-error samples* are unperturbed prefixes drawn from successful trajectories before any injected fault. *Recoverable-error samples* are perturbed prefixes whose rollout still succeeds under π. *High-Risk samples* are perturbed prefixes whose rollout fails under π. We map the first two sample types to $y = 0$, since the task remains solvable under the policy, and map high-risk samples to $y = 1$, since continuation from the perturbed trace leads to task failure in rollout.

Training Objective. The dataset exhibits significant class imbalance: most intermediate steps contain no error or have recoverable errors, while high-risk errors are sparse. To ensure the model learns to identify rare but critical failures, we employ a class-weighted focal loss [14]:

$$\ell_{focal} = -w_i \cdot (1 - p_t)^\gamma \log(p_t),$$

where p_t is the predicted probability for the true class, γ is the focusing parameter, and w_i is a sample-specific weight based on its error type (`recoverable`, `high-risk`, `no error`). Concretely, for each training sample, we assign $w_i = \beta \cdot \frac{\text{freq}_{\text{all}}}{\text{freq}_i}$, where freq_i is the frequency of that sample's error type in the training set, freq_{all} is the mean frequency across all error types, and β is a scaling hyperparameter tuned on the validation set. This inverse-frequency weighting ensures that the model gives sufficient importance to rare high-risk patterns while still learning from common recoverable cases. Hyperparameters & implementation details are in Section 10.

4 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 12.

Proposition 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 12, the risk is bounded with high probability as:

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

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 $\mathcal{O}\Big(\sqrt{\frac{N}{B}}\Big) + \mathcal{O}\Big(\frac{1}{\sqrt{N}}\Big)$. 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 $\mathbb{E}_y[\nabla_\theta \ell(f_\theta(\tau_{\le t}), y)] = \nabla_\theta \mathbb{E}_y[\ell(f_\theta(\tau_{\le t}), y)]$, ensuring SGD convergence under standard conditions [19, 3]. 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 16 (Section 14), 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. Under a fixed budget, allocating rollouts to more distinct states also broadens the observed coverage of task domains, tools, and action patterns (Table 17), while most single-rollout labels remain stable under repeated rollouts (Appendix 14).

5 Experimental Setup

Public Dataset We evaluate on the *"Who and When"* benchmark [36], a challenging dataset designed for fault localization in agentic traces. It aggregates and annotates traces from two popular open-ended agent benchmarks: (i) *GAIA* [17] and (ii) *AssistantBench* [33]. We use its Hand-Crafted subset, which consists of 58 queries (30 from GAIA, 28 from AssistantBench) that produce incorrect outputs when executed by *Magentic-One* [6], a state-of-the-art multi-agent system powered by *GPT-4o*. Expert annotators then reviewed the execution traces to identify the *Root Cause Step* t_{cause}, defined as the earliest irrecoverable error that guarantees downstream failure. Following the protocol in our data generation procedure, we cast this as binary classification: for every trace with an identified cause at t_{cause}, we label the partial trace $\tau_{\le t_{\text{cause}}}$ as $y = 1$ (High-Risk), and all preceding partial traces $\tau_{\le t}$, where $t < t_{\text{cause}}$, as $y = 0$ (Safe-to-continue).

Our Synthetic Dataset: Table 1 summarizes the statistics of data generated using the procedure in Section 3. Since recoverability is policy-conditioned, we construct data using the same agent policy family used for the corresponding task category. For **WebQA**, we use *Magentic-One* [6] with `gpt-4o`, using GAIA queries excluding those in the *"Who and When"* benchmark, together with 1000 HotpotQA queries. This choice keeps the setup aligned with the public *"Who and When"* benchmark. For **Coding**, we use MBPP traces generated with *Magentic-One* and `gpt-4o`, representing short-horizon program-synthesis tasks. We use the MBPP training split (374 tasks) to construct synthetic training data for the forecaster, and reserve the MBPP validation split (90 tasks) for downstream evaluation. Unlike the standard MBPP protocol, where test assertions

Table 1: Overview of sample distribution across the training and test sets. "Safe" refers to no-error or recoverable-error prefixes; "High-risk" refers to prefixes whose continuation leads to task failure. Test* is human-annotated Who&When [36]

Category	Split	Total	Safe	High-Risk
WebQA	Train	3958	3562	396
	Test	480	430	50
	Test* [36]	937	879	58
Coding	Train	3248	2816	432
	Test	630	583	47
Agentic	Train	1604	1251	353
	Test	410	315	95

are visible in the prompt, the agent receives only the natural-language problem statement; reference tests are hidden and used only for final evaluation, preventing the agent from tailoring its solution to the assertions. This accounts for the lower absolute accuracy in Table 3; results under the standard test-visible protocol are in Appendix 14. For **Agentic** tasks, we follow the *EnterpriseBench* evaluation setting and use a ReAct agent (with `GPT-5.1`). EnterpriseBench contains 500 tasks; we use 450 tasks for training-data construction and hold out 50 tasks for testing.

Baselines. We compare our forecaster against general-purpose LLMs prompted to act as online monitors. At every step t, models observe the partial trace $\tau_{\leq t}$ and predict whether the current trace will lead to failure. We evaluate: (i) *general-purpose closed-source models:* Gemini-2.5 Pro (Google), Claude-4.5 Sonnet (Anthropic), and GPT-4o (OpenAI); and (ii) *open-weights model:* Qwen-3-8B. Our method is a *Qwen-3-8B* model fine-tuned solely on the synthetic training data from the corresponding category, using the category-specific agent policy described above. All models are decoded with temperature 0; full inference parameters are listed in Table 5.

Metrics We evaluate performance using binary classification metrics computed at the step level: (i) *F1-Score (Weighted):* the harmonic mean of precision and recall, weighted by the support of each class (Safe-to-continue vs. High-Risk); and (ii) *Inference Cost:* the dollar cost to evaluate the full test set, based on standard API pricing per 1M tokens from `openrouter` (https://openrouter.ai/).

6 Experimental Results

We structure our evaluation around four research dimensions, via focused research questions (RQ), to validate both the efficacy and utility of our failure forecasting framework:

- **Forecasting Capability:** We benchmark our model's accuracy in detecting high-risk errors against general-purpose LLMs (**RQ1**) and examine its robustness when the underlying agent policy varies (**RQ6**, Section 14).
- **Downstream Utility:** We assess the practical value of forecasting by deploying our model as a gating mechanism for inference-time scaling, evaluating whether it can improve agent success rates on both held-out and unseen tasks (**RQ2**).
- **Data Generation Validity:** We validate our data synthesis pipeline by evaluating the efficiency of adaptive fault injection (**RQ3**) and plausibility of generated traces (**RQ4**, Section 14).
- **Problem Characterization:** We investigate intrinsic difficulty of failure forecasting through two analysis: (i) human study on distinguishing recoverable vs. high-risk errors in partial traces, (ii) analysis of geometric structure of partial trace embeddings to check separability (**RQ5**, Section 14)

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

Table 2: **Failure Forecasting F1 scores, inference costs, and uncertainty of gains over Qwen-3 8B.** "Safe" = Safe-to-continue class F1; "Risky" = High-Risk class F1; "W-Avg" = Weighted Average F1. The shaded uncertainty row reports Δ_f - Δ_b with 95% confidence intervals from 1000 bootstrap samples. Confidence intervals are shown for class-level F1 scores; W-Avg and Cost are therefore marked "–" in the uncertainty row. "Cost" denotes total forecasting inference cost in USD. Bold denotes the best score or lowest cost in each column, and underline denotes the second-best score.

	Who&When [36] **GAIA + AssistantBench**				**Our test GAIA + HotpotQA**				**Our test EnterpriseBench**				**Our test MBPP**			
Model	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]	–	–	+42.4 [+37.2, +49.5]	+3.7 [+1.0, +25.5]	–	–	+10.6 [+7.2, +14.0]	+1.52 [-0.6, +6.0]	–	–	+43.0 [+31.7, +45.0]	+10.0 [+5.4, +11.2]	–	–

We evaluate our fine-tuned Qwen-3-8B forecaster against general-purpose LLMs prompted as zero-shot evaluators (with CoT) on both the human-annotated *"Who and When"* benchmark and our own held-out synthetic test sets (Table 2). On "Who and When", our forecaster significantly outperforms large general-purpose LLMs. It achieves the highest Weighted F1-score (78.01), surpassing GPT-4o (68.66) and drastically outperforming base Qwen-3-8B model (8.78). While general-purpose models like GPT-4o and Gemini-2.5 Pro show competitive or slightly higher F1-scores for 'High-risk' class in isolation, their overall 'Weighted Avg' F1-scores are lower. This is due to their imbalanced predictive

behavior where they frequently misclassify safe steps. Our model maintains strong precision on 'Safe-to-continue' steps (94.93%) while retaining reasonable recall on 'High-Risk' errors (43.86%) (Table 10a). Crucially, we achieve superior performance at an inference cost of \$0.13, $\approx 100\times$ lower than Claude-4.5 Sonnet. Similarly, on our held-out test sets of other benchmarks, our forecaster demonstrates even more pronounced gains outperforming general-purpose LLMs. The forecaster also generalizes beyond the Error Bank: on Who&When failures judged to fall outside all Error Bank categories, it reaches 85.7 weighted F1 versus 83.3 for the best prompted model (Appendix 14).

Takeaway: *Training on synthetic data yields a forecaster that generalizes well to human-labeled failures, and at significantly lower inference cost compared to larger, general-purpose LLMs.*

RQ2: *Can a learned forecaster be used to selectively allocate inference-time scaling compute, including on unseen tasks, to mitigate predicted failures?*

While detecting errors is valuable, the ultimate utility of a forecaster lies in whether it improves downstream task performance. We therefore use forecaster as a gating mechanism for inference-time scaling, invoking additional test-time compute only when the current partial trace is predicted to fail.

We use a *Greedy Budgeted Intervention* protocol. Let $\mathcal{M}_{\text{scaling}}$ be an inference-time scaling module and assign each query a budget $K = 5$ calls. For a trace τ, at each step t, the agent first generates an action $a_t \sim \pi(\cdot \mid \tau_{<t})$. The forecaster then evaluates the resulting partial trace, $\hat{p} = f_\theta(\tau_{\leq t})$. If failure is predicted and the remaining budget $k > 0$, we discard a_t and invoke the scaling module to generate an alternative action, $a'_t \leftarrow \mathcal{M}_{\text{scaling}}(\tau_{<t})$. We update the trace with a'_t and decrement the budget, $k \leftarrow k - 1$. Otherwise, we proceed with original action a_t. This tests whether the forecaster can target intervention compute to high-risk decision points instead of applying scaling uniformly. The full procedure is given in Algorithm 2, and the agent system, backbone LLM, and scaling strategy for each benchmark are summarized in Table 6 (Appendix 11).

Table 3: **Inference Scaling Performance.** We compare accuracy and total inference cost when different forecasters are used to gate inference-time scaling. For short-trace tasks, we use Best-of-N ($N = 3$), where predicted failures are typically locally repairable by resampling the next action. For long-trace coding tasks, we use Self-Refine, which allows the agent to revisit earlier decisions in the trace. Costs include the base agent run, forecasting, and any triggered scaling calls.

Dataset	Forecaster/Strategy	Accuracy	Inference Cost
Short-trace tasks *(10–30-step agent traces)*			
HotpotQA (Held-out)	None (No Scaling)	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 (No Scaling)	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 (No Scaling)	0.66	\$4.57
	Claude 4.5 Sonnet + BoN	0.84	\$9.23
	Qwen-3 8B + BoN	0.78	\$7.86
	Post-hoc targeted retry [16]	0.76	\$4.90
	Ours + BoN	0.82 ↑24.2%	\$7.83
GAIA & Ass. Bench (Who & When [36])	None (No Scaling)	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%	\$35.63
MBPP (Held-out)	None (No Scaling)	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-step agent traces)*			
SWE-Bench (Held-out)	None (No Scaling)	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

We instantiate $\mathcal{M}_{\text{scaling}}$ differently for short- and long-horizon tasks. For shorter-trace tasks (HotpotQA, MuSiQue, EnterpriseBench, Who and When, MBPP), which average roughly 10 - 30 agent steps, we use step-level Best-of-N generation with $N = 3$. In this setting, a forecaster trigger is often locally repairable by resampling the immediate continuation: for example, the agent may need to issue a better search query or choose a more relevant document. SWE-Bench is a harder long-horizon coding setting: unlike MBPP, which consists of short program-synthesis executed with *Magentic-One*, SWE-Bench requires repository-level issue resolution, for which we use *OpenHands* [28]. Its traces average roughly 60 - 65 steps, making synthetic rollout generation more expensive; hence we initialize from the MBPP-trained coding forecaster and further fine-tune on a smaller SWE-Bench-OpenHands synthetic dataset. Long-horizon coding tasks also require a different repair mechanism. A trigger at step t may indicate that ordinary continuation is unlikely to succeed, but the decision needing revision may have occurred much earlier, such as editing the wrong file. Therefore, instead of step-level Best-of-N resampling, we use Self-Refine for SWE-Bench. For short-horizon tasks, M_{scaling} samples N=3 alternative next actions and selects one using the agent's own backbone LLM;

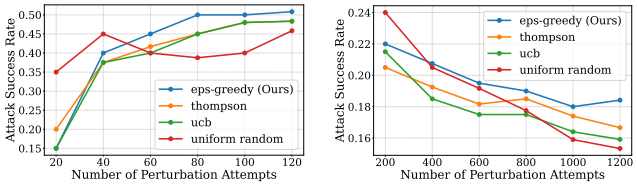
for SWE-Bench, Self-Refine generates a reflection and a revised action, and execution continues from the current state.

Table 3 shows that our forecaster is an effective cost-aware trigger for inference-time scaling across task regimes. On web and multi-hop QA tasks, it matches or exceeds stronger LLM-gated baselines: on HotpotQA (50 hard category tasks), it improves accuracy from 70% to 74%, while on unseen MuSiQue (50, 4-hop tasks) it improves accuracy from 12% to 24%, outperforming (or on par with) all LLM-gated baselines at a lower cost. On the public Who&When benchmark (58 failure trajectories from GAIA and AssistantBench), it improves accuracy from 10% to 14%, while remaining cheaper. On more agentic tasks our method remains competitive with state-of-the-art LLM gates while being substantially cheaper: on EnterpriseBench (50 tasks), it reaches 82% accuracy, close to Claude-gated scaling at lower cost. It also outperforms the post-hoc attribution-and-retry baseline (76%). Post-hoc repair is cheaper, but it acts only after the trajectory has completed. Online forecasting intervenes while the correct prefix is intact, before an erroneous tool call propagates, which matters when later actions are costly, irreversible, or user-visible. For coding, our method improves performance in both short- and long-horizon regimes. On MBPP val set (90 tasks), where local repair through Best-of-N is plausible, it improves accuracy from 47% to 49%. On SWE-Bench (100 tasks), it reaches 69%, within one point of Claude-gated Self-Refine while reducing total inference cost by more than $20\times$. In contrast, the base Qwen-3-8B gate often degrades performance despite similar forecasting cost, suggesting that the gains come from our learned policy-conditioned failure signal rather than simply adding a small model as a gate.

Takeaway: *The learned forecaster enables targeted inference-time intervention, outperforming LLM-gated scaling on web QA tasks and approaching state-of-the-art LLM gates on agentic and coding tasks at a fraction of the cost.*

***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.



(a) GAIA Dataset (b) HotpotQA Dataset

Figure 2: **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.

Across both GAIA and HotpotQA, ε-greedy consistently yields the highest ASR. On GAIA (Figure 2a), ASR increases over time with ε-greedy outperforming others. On HotpotQA (Figure 2b), 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 11 (Section 14), where bandit allocates nearly $7\times$ 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

Table 4: **Classification Performance ("Who and When" benchmark).** Comparison of F1 scores across classes for our method using Bandit selection vs Random (under same perturbation budget).

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*

to downstream forecasting performance. As shown in Table 4, 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.

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

7 Related Works

Recent advances in LLM agents have enabled complex multi-step reasoning and tool use [32, 21, 17, 33], yet failure rates remain high on challenging benchmarks. Prior work addresses agent failures through three main approaches: post-hoc failure analysis, step-level process supervision, and outcome-based value estimation. However, existing methods do not explicitly model or predict inevitable failure from partial execution traces. (i) *Post-Hoc Failure Analysis.* Studies examine failure patterns such as task disobedience and reasoning-action mismatches via manual analysis of completed traces [4, 5]. Automated approaches [34, 11] perform fault localization after task completion. DoVer [16] further intervenes at the attributed step and resumes execution; we compare against such a post-hoc attribution-and-retry baseline in Table 3. While informative, these methods do not support failure prediction during execution. (ii) *Process Supervision & Step-Level Judgments.* Process Reward Models provide step-level scores, typically trained via human annotation or outcome-based supervision [12, 25, 35]. They however do not predict global task reachability: locally incorrect actions may be recoverable, while plausible actions may irreversibly constrain future outcomes. (iii) *Outcome Prediction & Value Estimation.* Our problem relates to value function approximation in RL [23], where rollouts estimate future task success. We formalize Task Failure Forecasting as learning the complement of value function, using fault injection with rollout verification to obtain supervision for identifying high-risk states. Finally, fault injection has a long history in systems reliability [8]. We do not claim fault injection itself as novel; we adapt it to generate policy-conditioned recoverability supervision for forecasting failures in LLM-agent traces. A more detailed survey of works is in Section 13

8 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-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.

Within this setting, the forecaster is most reliable when the executing policy is fixed or changes moderately, and when the deployment task distribution stays close to the training distribution. It is robust to changes in backbone LLM, system prompt, and tool descriptions (Appendix 14). Recalibration is needed when a change materially alters the likelihood of recovery, e.g., new agent architectures, tool semantics or permissions, observation formats, or a persistent new class of failures. In practice, we recommend monitoring the high-risk false-negative rate on recent executions and, when it degrades, rollout-verifying new traces and combining them with targeted fault injection for incremental fine-tuning.

Second, injected faults are single-step perturbations, whereas natural failures may emerge through gradual multi-step degradation. Although the forecaster transfers to human-annotated and out-of-bank natural failures, the injected distribution is not matched to the natural one, and forecasting compound faults remains open. Third, single-rollout labels are noisy near the recoverability boundary (Appendix 14). Fourth, the human-annotated benchmark is small (58 traces), which we complement with rollout-verified test sets across three domains. Finally, the forecaster consumes a textual serialization of the trace; directly encoding structured or multimodal environment state (e.g., database states, logs, visual observations) is a promising direction.

References

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

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

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

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

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

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

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

[8] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer. Fault injection techniques and tools. *Computer*, 30(4):75–82, 1997.

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

[10] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157, 2024.

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

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

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

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

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

[16] M. Ma, J. Zhang, F. Yang, Y. Kang, Q. Lin, S. Rajmohan, and D. Zhang. Dover: Intervention-driven auto debugging for llm multi-agent systems. In *International Conference on Learning Representations*, volume 2026, pages 28059–28087, 2026.

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

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

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

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

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

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

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

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

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

[26] H. Vishwakarma, A. Agarwal, O. Patil, C. Devaguptapu, and M. Chandran. 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*, pages 9178–9212, 2025.

[27] P. Wang, L. Li, Z. Shao, R. Xu, D. Dai, Y. Li, D. Chen, Y. Wu, and Z. Sui. 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)*, pages 9426–9439, 2024.

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

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

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

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

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

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

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

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

[36] S. Zhang, M. Yin, J. Zhang, J. Liu, Z. Han, J. Zhang, B. Li, C. Wang, H. Wang, Y. Chen, 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*, 2025.

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

[38] C. Zhu, S. Hong, J. Wu, K. Chawla, C. Tang, Y. Yin, N. Wolfe, E. Babinsky, and D. Liu. 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. In particular, this appendix contains the following:

- **Proposed Algorithm**: Complete algorithmic description and visualization of our adaptive fault injection framework.
- **Forecasting Model Training Details**: Hyperparameters & implementation specifics for the forecasting model.
- **Inference-Time Scaling Details**: Implementation details and additional results corresponding to inference scaling experiments
- **Proposition Proof and Discussion**: Formal proof of the risk decomposition and extended discussion of the compute-coverage tradeoff.
- **Extended Related Works**: Comprehensive review of prior work on agent failures, process supervision, and value estimation.
- **Additional Results**: Continuation of research questions from Section 6, including bandit allocation analysis, trace quality evaluation, human annotation study, embedding analysis, and policy generalization experiments.
- **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.

9 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.

10 Forecasting Model Training Details

In this section, we continue the discussion of the training procedure introduced in Section 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 factor. This scaling was introduced to mitigate the vanishing gradient problem commonly observed when using focal loss[7]. 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.

Algorithm 1 Data Generation via Adaptive Fault Injection

Require: Successful trace $\tau^* = (o_0, a_0^*, \ldots, a_T^*)$, 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 $(\tau_{<t}^* \cup \{\tilde{a}_t\},\ y{=}1)$ to $\mathcal{D}$
    Update π_{bandit} with reward $r = 1$ (Failure-inducing perturbation)
  else
    Add $(\tau_{<t}^* \cup \{\tilde{a}_t\},\ y{=}0)$ to $\mathcal{D}$
    Update π_{bandit} with reward $r = 0$
  end if
end for
return Dataset $\mathcal{D}$

Training was conducted using Low-Rank Adaptation (LoRA) [9] 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.

Inference parameters. Table 5 lists the decoding parameters for all forecasters. Unlisted parameters use provider defaults.

Table 5: **Forecaster inference parameters.**

Model	Temperature	Top-p	Max output tokens
Qwen3-8B (vLLM)	0	1.0	1,000
Ours (vLLM)	0	1.0	1,000
GPT-4o	0	1.0	16,384
Gemini-2.5 Pro	0	0.95	65,536
Claude-4.5 Sonnet	0	0.999	64K

11 Inference-Time Scaling Details

Algorithm 2 details the Greedy Budgeted Intervention protocol used in RQ2, with $K{=}5$. Table 6 lists the agent configuration per benchmark.

Table 6: **Agent configuration for inference-time scaling experiments.**

Benchmark(s)	Agent system	Backbone	Trace length	Scaling
HotpotQA, MuSiQue	Magentic-One	GPT-4o	10–30	Best-of-N ($N{=}3$)
Who&When, MBPP	Magentic-One	GPT-4o	10–30	Best-of-N ($N{=}3$)
EnterpriseBench	ReAct	GPT-5.1	10–30	Best-of-N ($N{=}3$)
SWE-Bench	OpenHands	GPT-5.1-codex-mini	60–65	Self-Refine

Algorithm 2 Greedy Budgeted Intervention

Require: Agent policy π, forecaster f_θ, scaling module M_{scaling}, budget K
$k \leftarrow K$
while task not terminated **and** agent step budget not exhausted **do**
  $a_t \sim \pi(\cdot \mid \tau_{<t})$
  $\hat{y}_t \leftarrow f_\theta(\tau_{\leq t})$
  if $\hat{y}_t = \texttt{Incorrect}$ **and** $k > 0$ **then**
    Discard a_t; $a_t \leftarrow M_{\text{scaling}}(\tau_{<t})$
    $k \leftarrow k - 1$
  end if
  Append a_t to τ
end while

11.1 Cost and Latency Overhead

Table 7 reports the cost–performance trade-off of forecaster-gated scaling relative to no scaling, derived from Table 3. Aggregated over all benchmarks, total inference cost increases by $1.17\times$; larger multipliers occur only where the base run is very cheap (MBPP, EnterpriseBench). Table 8 reports wall-clock latency; Qwen3-8B and our forecaster run on 2 NVIDIA A40 GPUs.

Table 7: **Cost–performance trade-off** of Ours + BoN/Self-Refine versus no scaling.

Task	Acc. gain	Rel. acc. gain	Cost increase	Rel. cost increase	Cost multiplier
HotpotQA	+4%	+5.7%	+$2.51	+9.1%	1.09×
MuSiQue	+12%	+100.0%	+$2.02	+9.9%	1.10×
EnterpriseBench	+16%	+24.2%	+$3.26	+71.3%	1.71×
GAIA & AssistantBench	+4%	+40.0%	+$3.06	+9.4%	1.09×
MBPP	+2%	+4.3%	+$4.05	+129.4%	2.29×
SWE-Bench	+2%	+3.0%	+$1.69	+19.9%	1.20×

Table 8: **Mean wall-clock latency (seconds).** "Baseline": agent run without scaling. "With scaling": end-to-end latency with Best-of-N (EnterpriseBench) or Self-Refine (SWE-Bench). "Fc./query": cumulative forecaster latency per query. "Fc./call": latency of a single forecaster call.

Setting	Forecaster	Baseline	With scaling	Fc./query	Fc./call
Short horizon (EnterpriseBench)	Claude-4.5 Sonnet	10.89	31.71	7.21	4.51
	Qwen3-8B	10.89	25.91	1.71	1.07
	Ours	10.89	22.28	0.82	0.54
Long horizon (SWE-Bench)	Claude-4.5 Sonnet	1088.96	3392.01	152.18	12.27
	Qwen3-8B	1088.96	2195.75	31.15	6.92
	Ours	1088.96	3091.16	63.93	10.15

12 Proposition Proof and Discussion

Here we provide the complete proof of Proposition 1 from Section 4, 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 [18], 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 [19, 3].

13 Extended Related Works

This section expands the related work discussion from Section 7 and throughout the main paper, providing a comprehensive survey of prior work on LLM agent failures, process supervision, and outcome prediction. We position our work within this broader landscape and 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 [32, 21], with benchmarks exposing frequent long-horizon failures [17, 33]. Prior work largely studies failures post-hoc, via human annotation or fault localization [6, 36], which is ambiguous due to cross-step dependencies and error propagation in agent interaction graphs [4, 5]. Automated approaches such as AgentTracer [34] and AEGIS [11] 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 [13, 20, 22, 2, 25, 12, 35]. 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 [37]. 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 [36, 38], 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 [23], where Monte Carlo rollouts provide unbiased but high-variance estimates [19]. In language-based reasoning, rollout-based estimation is highly sensitive to policy quality, limiting its reliability in practice [37]. We formalize Task Failure Forecasting as approximating the complement of the value function and use rollout-verified fault injection to obtain unambiguous supervision.

Table 9: **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.

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**

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.

14 Additional Results

This section extends the experimental evaluation from Section 6 with additional research questions and analysis that provide deeper insights into our method's performance and characteristics.

In Section 6 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).

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

Extending our analysis in Section 6 for RQ3 in the main paper, Table 11 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.

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 two 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

Table 10: **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) EnterpriseBench and (d) MBPP. 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.

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) [36]

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	0.21	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

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 the same test-set traces used in RQ5 (composition given there); Study 2 used only the perturbed traces from this set.

Takeaway: *Human evaluation in Table 12 shows that our data generation pipeline produces coherent, plausible traces with fault injections that faithfully reflect the intended error 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. Annotators were given a description of the agent system and its backbone LLM, since recoverability is policy-conditioned; the study measures how hard forecasting is for human observers, not ground-truth labels. Table 13 shows that while annotators reliably detect

Table 11: **Bandit Attack Allocation Analysis.** Performance on top-3 and bottom-3 failure categories, ranked by Random baseline ASR. Darker cells indicate more attacks.

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 12: **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.

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%

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.

Table 13: **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.

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

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 3). 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

Table 14: **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.

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**

of action itself, but rather a latent property of how environment state evolves and whether policy π can recover from that state.

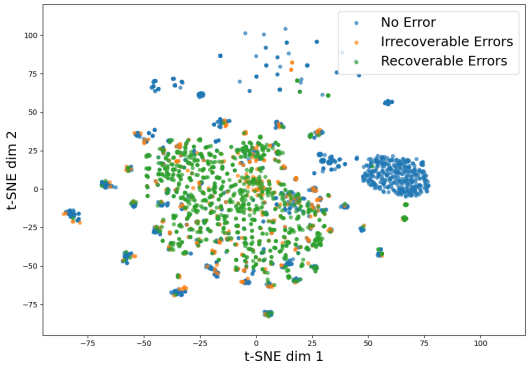


Figure 3: **t-SNE Visualization of Last Agent Action Embeddings.** We visualize the representations of the last action across traces from our generated training set, color-coded by their ground-truth error status. While benign actions ("No error") cluster, recoverable and high-risk errors overlap significantly. This shows detecting irrecoverability requires analyzing the agent trace's causal future, not just the action's content.

Takeaway: *Error severity is not linguistically or semantically distinct from action content alone. Thus, accurate forecasting requires modeling the agent-environment interaction 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 14 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).

Shifts in model family, prompt, and tools. We further evaluate the EnterpriseBench forecaster, trained on a GPT-5.1 ReAct agent with standard prompts and tools, under three kinds of policy shift (Table 15). For backbone shifts, we generate a test set per backbone with our pipeline (50 tasks, 3 perturbation sites per successful trace). For prompt and tool shifts, we keep GPT-5.1 and the same 50 tasks, and either add an explicit verification instruction before termination (*strict verification*), remove guardrails from the standard prompt (*minimal*), or deterministically paraphrase tool descriptions (e.g., "Retrieve"→"Fetch", "Create"→"Add", "Deletes"→"Removes"). Our forecaster consistently outperforms the base model under all shifts, including across model families.

Table 15: **Robustness to policy shift on EnterpriseBench.** The forecaster is trained on a GPT-5.1 ReAct agent with standard prompts and tools, and evaluated under shifts in backbone LLM, prompt, and tool descriptions. Support is reported as Safe/High-Risk partial traces. ↑ denotes the absolute W-Avg F1 gain over Qwen3-8B.

Policy shift	Support	Forecaster	Safe F1	Risky F1	W-Avg F1
Backbone LLM shift *(standard prompt & tools)*					
GPT-5.2 Chat	59/8	Qwen3-8B	62.79	**33.33**	59.27
		Ours	**74.75**	33.00	**69.76** ↑10.5
Claude-4.5 Sonnet	55/6	Qwen3-8B	62.65	20.51	58.51
		Ours	**83.67**	**33.33**	**78.72** ↑20.2
Prompt / tool shift *(GPT-5.1 backbone)*					
None (training policy)	73/11	Qwen3-8B	59.81	29.51	55.84
		Ours	**79.03**	**40.91**	**74.04** ↑18.2
Strict verification prompt	70/23	Qwen3-8B	66.06	51.95	62.57
		Ours	**78.05**	**57.14**	**72.88** ↑10.3
Minimal prompt	73/17	Qwen3-8B	67.86	47.06	63.93
		Ours	**78.05**	**52.63**	**73.25** ↑9.3
Paraphrased tool descriptions	59/12	Qwen3-8B	64.37	43.64	60.86
		Ours	**78.79**	**51.16**	**74.12** ↑13.3

Takeaway: *Our forecaster generalizes to moderate policy shifts (changes in backbone LLM, including across model families, system prompt, and tool descriptions), consistently outperforming the base model. It is nonetheless trained on a single policy; training policy-agnostic forecasters from multi-policy data remains important future work.*

RQ7: *Under the same compute budget (number of rollouts), does it help prioritizing diversity over label precision as argued in Section 4?*

In Section 4, 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 16 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

Table 16: **Empirical Validation of Diversity vs. Precision Tradeoff.** 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.

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*

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.

Takeaway: *Empirical results validate our theoretical analysis from Section 4: 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.*

Does more states imply broader coverage? To check that state count translates into coverage, we vary the allocation $S \times N$ under a fixed budget B=100 over the attacked partial states of EnterpriseBench. We measure coverage before any resumed rollout via tool domains (CRM, HR, IT, etc.), unique tools, and unique tool-call bigrams/trigrams (Table 17). Moving from S=5 to S=100 increases domain coverage from 4 to 7, unique tools from 8 to 35, and trigram diversity from 7 to 38. Spending rollouts on fewer states sharpens their labels but exposes the forecaster to few new domains, tools, or action patterns.

Table 17: **State coverage under a fixed rollout budget** ($B = S \cdot N = 100$, EnterpriseBench).

States S	Rollouts/state N	Domains	Unique tools	Tool bigrams	Tool trigrams
100	1	7	35	48	38
50	2	7	31	39	29
25	4	7	26	30	20
20	5	7	24	26	19
10	10	5	13	14	9
5	20	4	8	9	7

How stable are single-rollout labels? We re-ran 3 rollouts on each of 20 injected states under the same ReAct policy. 13/20 states ($\sim$65%) yield identical labels across rollouts, while 7/20 vary, with mean within-state variance 0.077. Most labels are thus stable, but some injected states lie near the recoverability boundary, consistent with treating single-rollout labels as noisy but unbiased samples (section 4).

RQ8: *Does the forecaster generalize to natural failures outside the Error Bank?*

To identify out-of-bank failures in Who&When, two LLMs (GPT-4o and Claude-4.5 Sonnet) independently judged whether each human-annotated root-cause error matches an Error Bank category. Errors both judged as unmatched form the out-of-bank subset; disagreements were resolved by human

adjudication. As Table 18 shows, our forecaster achieves the highest weighted and High-Risk F1 on this subset.

Table 18: **Forecasting on out-of-bank natural failures** (Who&When).

Model	Safe F1	Risky F1	W-Avg F1
Claude-4.5 Sonnet	65.5	17.4	61.8
Gemini-2.5 Pro	86.1	28.6	82.1
GPT-4o	88.5	0.0	83.3
Qwen3-8B	54.5	19.4	52.1
Ours	**88.9**	**42.9**	**85.7**

Takeaway: *Supervision from adaptive fault injection transfers to naturally occurring failures, including those outside the Error Bank's categories.*

RQ9: *Is fault injection a better source of supervision than attribution-derived labels?*

We train Qwen3-8B with identical hyperparameters on labels from LLM-based root-cause localization of naturally failed EnterpriseBench traces (1,883 samples: 1,674 Safe, 209 High-Risk). We evaluate it both as a forecaster on our test set and as a gate for Best-of-N, alongside the post-hoc targeted-retry baseline from Table 3. Attribution-derived labels do not improve over the base model, while fault-injection supervision yields the best forecaster and the best downstream accuracy (Table 19).

Table 19: **Fault injection vs. attribution-derived supervision** (EnterpriseBench). *Forecasting* reports step-level F1 of each forecaster on our held-out test set (*Safe F1, Risky F1, W-Avg F1*). *Downstream* reports task accuracy when the forecaster gates Best-of-N (N=3) inference-time scaling. "–" marks strategies that use no forecaster (no scaling, and the post-hoc baseline, which attributes failure only after the episode completes), so forecasting F1 is undefined for them.

Forecaster / Strategy	Safe F1	Risky F1	W-Avg F1	Task Accuracy
None (No Scaling)	–	–	–	0.66
Post-hoc targeted retry	–	–	–	0.76
Qwen3-8B (base) + BoN	66.4	40.5	60.4	0.78
Qwen3-8B (attribution labels) + BoN	65.0	40.3	59.0	0.73
Ours + BoN	**77.0**	**42.1**	**68.9**	**0.82**

MBPP with visible tests. In our main MBPP setting, reference tests are hidden from the agent (Section 5). Repeating the experiment on the same 90 validation tasks with tests provided in the prompt raises the no-scaling baseline from 47% to 86.67%, confirming that the lower accuracy stems from the hidden-test protocol. Even in this setting, forecaster-gated Best-of-N improves accuracy to 91.11%, outperforming Claude-4.5 Sonnet- and Qwen3-8B-gated Best-of-N (both 87.78%).

15 Prompts

This section provides the complete prompt templates used throughout our data generation pipeline (Section 3.1) and evaluation procedures (Section 6). 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}
```

16 Human Evaluation Details

This section provides additional details on the human evaluation studies reported in RQ4 and RQ5. Figure 4 and Figure 5 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 12 and Table 13.

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 the RQ5 trace set in a "blind" mode (Figure 4), where they were unaware whether a trace was original or synthetically perturbed. Annotators were provided with the agent system description and backbone LLM for each trace. They were instructed to review the agent execution 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 the perturbed traces from this set (Figure 5). 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.

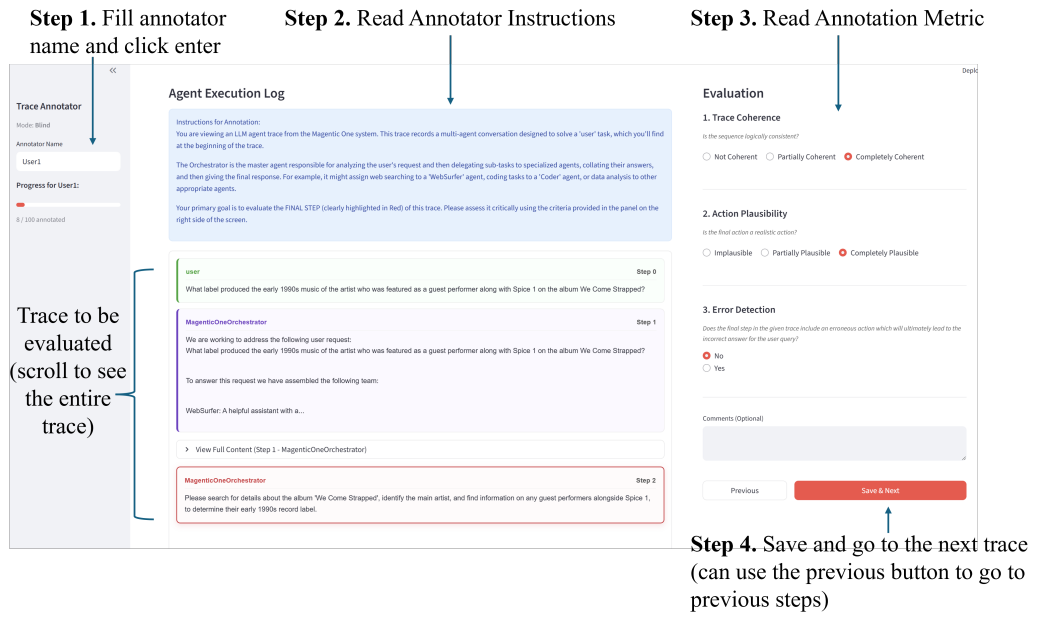


Figure 4: **Data Annotation Application View for Study 1**

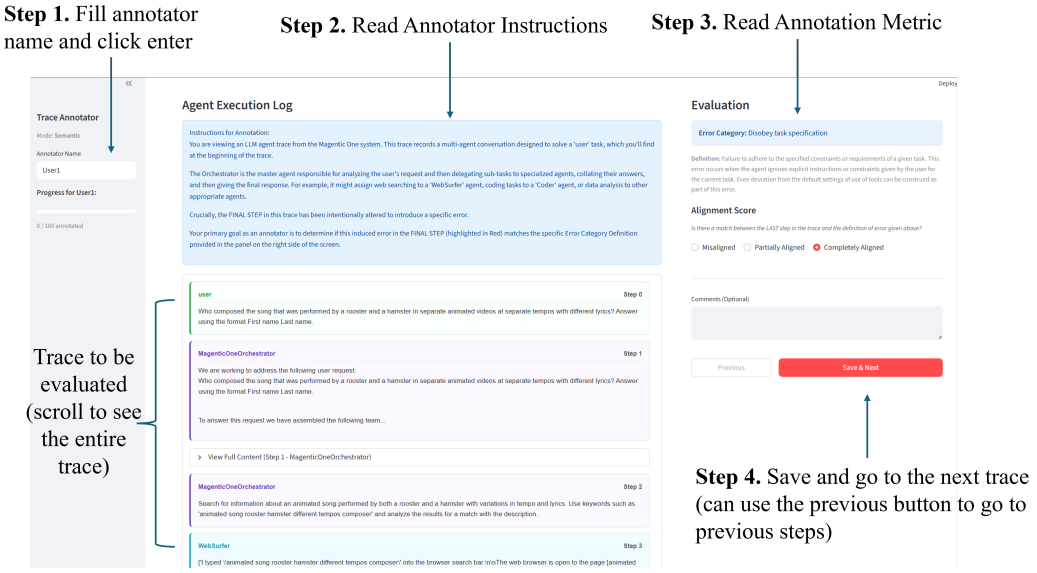


Figure 5: **Data Annotation Application View for Study 2**

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims made in the introduction (contribution list at the end of the section) are validated in the experimental Section 6.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See Section 8

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.

- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Section 4 in the main paper, and Section 12 in appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: See Section 5, and Section 10.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.

- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Fully documented and anonymized code attached in the zip file.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: See Section 5 and Section 10.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: See Table 2 for bootstrap confidence intervals of main forecasting results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: Appendix Section 10 reports that fine-tuning used four NVIDIA A40 GPUs for four epochs with an effective batch size of 160 and approximately eight hours of training time. The paper also reports inference and data-generation costs in Tables 2, 3, and 9, including LLM call counts and token statistics for pipeline components

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Experimentation is mostly performed on open benchmark datasets (see Section 5) which are widely acceptable. We have followed all the necessary code of ethics to conduct the research.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: There is no direct societal impact of the work.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper does not have such risks

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Properly acknowledged the assets used.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The details of dataset generation and statistics is given in Section 3 and Section 5 and details of model training is provided in Section 3 and Section 10. The code attached as zip is anonymized and well documented.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [Yes]

Justification: Screenshots and details are in Appendix 16.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: No such risks exist which the participants are exposed to.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs have been used in the methodology as detailed in Section 3, and Section 15 provides the prompts for the same.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.