LEARNING FROM SUCCESS: ONLINE EARLY FAILURE DETECTION FOR AGENTIC SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based agent systems are increasingly deployed for complex tasks, yet their execution success rates remain limited, making early failure detection critical for raising alerts or intervening in erroneous executions before completion. Existing studies primarily focus on post-hoc failure attribution, which requires complete execution trajectories and forgoes the opportunity to intervene when the task is still running, while the few efforts on early failure detection typically rely on supervised learning from failure trajectories with step-level error annotations. In practical deployments, however, annotating a sufficient number of failure trajectories for supervised failure detection is costly, whereas successful trajectories are naturally accumulated through routine execution. This motivates an important yet underexplored setting: safe-only early failure detection, where an effective monitor identifies failures online by learning exclusively from successful execution trajectories. We introduce a simple framework for this setting, grounded in the insight that successful trajectories provide complementary evidence about both how an execution should evolve and whether the current transition is consistent with previously observed behaviors. Accordingly, our framework combines a safe-dynamics predictor that models the temporal evolution of successful trajectories with a non-parametric safe transition memory that evaluates contextual consistency against historical safe transitions. When a limited number of unsafe trajectories become available, our framework naturally incorporates them as failure memory through causal retrieval and safe-unsafe transition contrast, without additional retraining or fine-tuning. Extensive experiments demonstrate that our framework achieves strong online failure detection performance under the safe-only setting and can further leverage limited failure supervision to approach the performance of fully supervised detectors while requiring fewer annotated failure trajectories.

1 INTRODUCTION

Large language models (LLMs) equipped with sophisticated harness and external tools have rapidly evolved into agentic systems that plan, act, and update their decisions through interaction (Yao et al., 2023; Schick et al., 2023). Multi-agent systems (MAS) further orchestrate specialized roles through inter-agent communication and structured workflows (Li et al., 2023; Hong et al., 2024; Wu et al., 2024). These systems increasingly operate in complex coding (Jimenez et al., 2024; Wang et al., 2025), scientific (Schmidgall et al., 2025), and web environments (Zhou et al., 2024; Mialon et al., 2024), where an execution may span many steps or agent interactions. However, representative evaluations (Zhang et al., 2025b) continue to reveal substantial reliability gaps, making execution failures a critical obstacle to dependable deployment.

In response, failure analysis and *failure attribution* for agentic execution trajectories have received increasing attention (Cemri et al., 2025; Deshpande et al., 2025; Zhang et al., 2025b; 2026b; Zhu et al., 2025). Given a failed trajectory, attribution methods aim to identify the step(s) and agent(s) responsible for the failure. Although valuable for debugging and recovery, post-hoc diagnosis is performed after execution has terminated, forfeiting the opportunity to alert or intervene while the task is still being executed. By the time diagnosis begins, the failed trajectory has already consumed computational, environmental, or external resources, and failures in deployed systems may produce

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

operational risks or irreversible side effects. This motivates *online early failure detection* [1], where a monitor evaluates each newly observed execution step using only the current trajectory prefix and decides whether the execution should continue or an alarm should be raised.

Existing early failure detectors naturally benefit from failed executions. They rely on decisive-error annotations, explicitly constructed unsafe trajectories, or internal representations associated with successful and failed episodes (Zhang et al., 2025b; 2026b;a). For example, AgentForesight-7B is trained through a two-stage reinforcement-learning procedure using safe and unsafe trajectories (Zhang et al., 2026a). Such supervision can be effective, but collecting failed trajectories and annotating their decisive error steps is costly, particularly for long-horizon tasks. This substantially limits the scalability of supervised failure detection in real-world deployments. In contrast, successful trajectories are naturally accumulated through routine executions of agent systems. Moreover, agent failures are heterogeneous and open-ended, spanning reasoning, planning, memory, action, verification, and coordination errors, making exhaustive coverage of future failure patterns difficult. *This motivates a complementary perspective: instead of attempting to enumerate how agents may fail, learn how successful execution should evolve and detect departures from that reference.*

We therefore study a more practical setting: *safe-only early failure detection*, where the monitor learns exclusively from successful execution trajectories and audits an unfolding execution online. We argue that successful executions provide evidence about how an execution should evolve. Given a task q and an execution prefix $x_{1:t-1}$, successful trajectories provide observations of the conditional safe execution process

$$p_{\text{safe}}(x_t \mid q, x_{1:t-1}).$$

A newly observed step that is incompatible with this expected evolution provides evidence that the execution is departing from successful behavior. Importantly, such compatibility is context dependent: the same action may be valid in one execution history but erroneous in another.

Based on this perspective, we propose **Learning from Success (LFS)**. LFS combines two complementary sources of safe evidence. A parametric safe-dynamics predictor models global temporal regularities of successful execution, while a non-parametric safe transition memory retrieves concrete successors that followed similar historical contexts. Their combination evaluates whether a newly observed step is consistent with both learned safe evolution and locally relevant successful transitions. When limited failed trajectories become available, LFS further incorporates their decisive transitions as a lightweight failure memory. These failed examples provide additional contrastive evidence without retraining or fine-tuning the safe monitor, yielding a natural adaptation from safe-only to failure-assisted early detection. Our contributions are summarized as follows:

- We propose LFS, which combines learned safe execution dynamics with non-parametric safe transition memory to capture complementary global temporal regularities and local contextual consistency.
- When limited failed trajectories become available, LFS can naturally incorporate them as contrastive transition memory without additional model training.
- Extensive experiments show that LFS provides strong online failure detection under the safe-only setting and can further exploit limited failure supervision to approach fully supervised detectors while using substantially fewer annotated failed trajectories. Our project is available at code.

2 Related Work

Multi-agent systems increasingly execute long, tool-augmented workflows involving reasoning, communication, and environment interaction (Yao et al., 2023; Schick et al., 2023; Li et al., 2023; Hong et al., 2024; Wu et al., 2024). As these trajectories become longer and more complex, recent work has moved beyond final-task evaluation toward failure analysis and attribution, including failure taxonomies (Cemri et al., 2025; Deshpande et al., 2025; Zhu et al., 2025), localization of responsible agents or decisive error steps (Zhang et al., 2025b; 2026b; 2025a), and intervention-based diagnosis (Ma et al., 2026). These approaches are largely post-hoc, whereas online monitoring requires decisions from the currently observed trajectory prefix. For online early failure prediction, AgentForesight (Zhang et al., 2026a) directly studies causal early detection but relies on supervised

[1] Here, *early* means detecting a failure during execution and before the trajectory is completed; it does not require forecasting the failure before the decisive erroneous step becomes observable.

safe and failed trajectories with decisive-error annotations. The ATS (Dubey, 2026) instead adopts one-class monitoring from healthy execution telemetry. The OAT (Yeh et al., 2026) learns the latent flow of successful trajectories for unsupervised failure attribution, but operates on completed trajectories. A detailed discussion of related work is provided in Appendix A.

3 Problem Formulation

Let q denote a task and $\tau = (q, x_1, \ldots, x_T)$ its observable execution trajectory, where each x_t may contain the acting agent's role, action or tool invocation, and observable content. An outcome function $Y(\tau) \in \{0, 1\}$ determines whether the completed execution succeeds. We call τ *safe* [2] if $Y(\tau) = 0$ and *unsafe* otherwise. For an unsafe trajectory with step-level annotations, t^* denotes the earliest annotated decisive failure step. Online early failure detection is prefix restricted. After observing step t, an auditor f_ϕ estimates a failure risk

$$r_t = f_\phi(q, x_{1:t}), \tag{1}$$

without access to future steps $x_{t+1:T}$. Unlike post-hoc attribution, which can inspect the completed failed trajectory, the auditor must make its decision while execution is still unfolding.

In conventional supervised early failure detection, the auditor f_ϕ can be trained using both safe and failed trajectories. In the *safe-only* setting studied in this work, its parameters are learned exclusively from successful executions $\mathcal{D}_{\text{safe}}$:

$$\phi^* = \arg\min_\phi \mathbb{E}_{\tau_i \in \mathcal{D}_{\text{safe}}} \mathcal{L}(f_\phi, \tau_i), \tag{2}$$

where $\mathcal{L}$ denotes a training objective. Failed trajectories, when available in our failure-assisted extension, are used only as non-parametric retrieval memory and never to update the safe monitor.

4 LFS: Learning from Success

Online failure detection repeatedly audits an execution as new steps arrive. Repeatedly prompting a generative LLM over the growing trajectory incurs substantial autoregressive inference cost. LFS instead operates in a compact execution-representation space: the task query and each newly observed step are encoded once, after which lightweight prediction, retrieval, and scoring are performed without text generation. Our method estimates whether the current step is compatible with successful execution through two complementary views. First, a parametric safe-dynamics predictor estimates the next-step representation expected from the query and execution history. Second, a non-parametric safe transition memory retrieves concrete successors that followed similar contexts in historical safe trajectories. Once the current step is observed, a scoring function compares it with both sources of safe expectation and produces an online risk score. A framework overview is shown in Figure 1.

4.1 Learning Safe Execution Dynamics

A raw agentic step can contain inter-agent messages, reasoning, tool calls, and tool outputs. Applying a prompt-based auditor directly to the growing raw trajectory at every step would repeatedly invoke expensive generative inference; recent safe-only attribution results likewise identify frontier-model inference as a major latency and token-cost bottleneck (Yeh et al., 2026). Formally, let $E(\cdot)$ be a fixed execution encoder with a pooling strategy based on a pretrained LLM. At the beginning of task execution, we obtain the task query representation $\mathbf{z}_q = E(q) \in \mathbb{R}^d$. At step t, E maps the individual steps into fixed-dimensional vectors,

$$\mathbf{z}_t = E(x_t) \in \mathbb{R}^d, \quad t = 1, 2, \cdots, T. \tag{3}$$

Successful executions provide observations of how an execution can evolve while remaining compatible with task success. We model this regularity through a causal predictor g_θ

$$\hat{\mathbf{z}}_t = g_\theta(\mathbf{h}_{t-1}, \mathbf{z}_{t-1}), \quad \mathbf{h}_{t-1} \in \mathbb{R}^d, \tag{4}$$

where $\mathbf{h}_{t-1}$ summarizes the preceding execution history and $\mathbf{z}_0$ represents the task query for the first prediction. The $\hat{\mathbf{z}}_t$ estimates the representation of the current step from the query and preceding steps. The essential object is the conditional next-step prediction.

[2] Throughout this paper, *safe* refers specifically to verified task success under the corresponding task objectives, rather than the broader notion of safety associated with alignment or harmlessness.

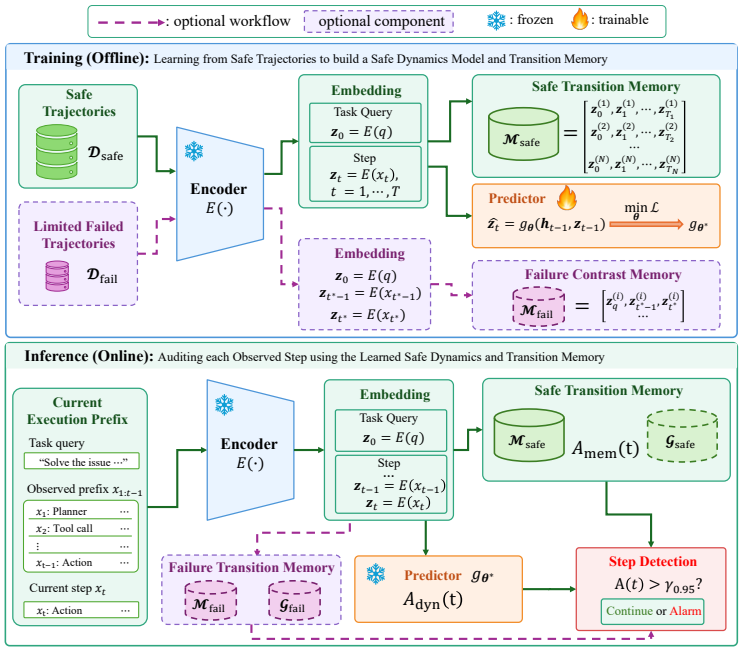

Figure 1: Overview of the LFS framework. Optional workflows depend on the availability of failure trajectories.

Let $\mathcal{D}_{\text{safe}}^{(\text{train})}$ denote the training set with N_{safe} safe trajectories. Given a discrepancy measure ℓ_{dyn}, the predictor is learned by

$$\theta^* = \arg\min_{\theta} \frac{1}{N_{\text{safe}}} \sum_{\tau_i \in \mathcal{D}_{\text{safe}}^{(\text{train})}} \frac{1}{T_i} \sum_{t=1}^{T_i} \ell_{\text{dyn}} \left(g_\theta(\mathbf{h}_{t-1}^{(i)}, \mathbf{z}_{t-1}^{(i)}), \mathbf{z}_t^{(i)} \right). \tag{5}$$

Averaging within each trajectory before averaging across trajectories prevents long executions from dominating the objective. During inference, for a new task $q^{(\text{new})}$, after $\mathbf{z}_t^{(\text{new})}$ is observed, deviation from the predicted safe evolution provides parametric anomaly evidence:

$$A_{\text{dyn}}(t) = \ell_{\text{dyn}}(\mathbf{z}_t^{(\text{new})}, \hat{\mathbf{z}}_t^{(\text{new})}), \tag{6}$$

where ℓ_{dyn} measures incompatibility of context in the representation space. A large value indicates that the observed step is difficult to explain under dynamics learned from successful executions.

4.2 Conditional Safe Transition Memory

A parametric predictor summarizes regularities shared across safe trajectories, but a context may admit several distinct successful continuations. A single prediction can blur these alternatives, particularly when safe execution is multi-modal. We therefore retain safe transitions themselves as a complementary non-parametric estimate of conditional safe behavior. No additional procedure is required to construct the safe memory; it is naturally organized as

$$\mathcal{M}_{\text{safe}} = \left\{ [\mathbf{z}_0^{(i)}, \mathbf{z}_1^{(i)}, \cdots, \mathbf{z}_{T_i}^{(i)}] : \tau_i \in \mathcal{D}_{\text{safe}}^{(\text{train})}, \mathbf{z}_0^{(i)} = \mathbf{z}_q^{(i)} \right\}, \tag{7}$$

where $\mathbf{z}_q^{(i)}$ and $\mathbf{z}_t^{(i)}$ are both byproducts of the training procedure of the fixed encoder $E(\cdot)$.

During inference, for a new task $q^{(\text{new})}$, before the current step $x_t^{(\text{new})}$ is observed, we use the available local context representation $\mathbf{z}_{t-1}^{(\text{new})} = E(x_{t-1}^{(\text{new})})$ to retrieve the K most similar historical contexts from the safe memory:

$$\mathcal{N}_K^{\text{safe}}(t) = \underset{(\mathbf{z}_j^{(i)}, \mathbf{z}_{j+1}^{(i)})}{\text{TopK}} \left[\text{sim} \left(\mathbf{z}_{t-1}^{(\text{new})}, [\mathcal{M}_{\text{safe}}]_j^{(\tau_i)} \right) \right], \qquad j \in \{0, 1, \ldots, T_i - 1\}, \tag{8}$$

where $\text{sim}(\cdot, \cdot)$ denotes the similarity measure, and $[\mathcal{M}_{\text{safe}}]_j^{(\tau_i)}$ denotes the step embedding $\mathbf{z}_j$ stored from safe trajectory τ_i. For each of the retrieved top-k contexts $\mathbf{z}_j$, we further obtain its corresponding successor embedding $\mathbf{z}_{j+1}$ from the same safe trajectory and also include these successor embeddings

in $\mathcal{N}_K^{\text{safe}}(t)$. These successor embeddings therefore represent the observed next states following historical safe contexts that are most similar to the current execution context, providing a retrieval-based reference for the expected safe evolution of the ongoing trajectory.

The successor embeddings associated with these retrieved contexts collectively form a conditional safe support set. Once $x_t^{(\text{new})}$ is observed, we obtain its embedding $\mathbf{z}_t^{(\text{new})}$ and measure its incompatibility with this support set as

$$A_{\text{mem}}(t) = \text{Agg}\left(\left\{d_{\text{mem}}\left(\mathbf{z}_t^{(\text{new})}, \mathbf{z}_{j+1}\right) : (\mathbf{z}_j, \mathbf{z}_{j+1}) \in \mathcal{N}_K^{\text{safe}}(t)\right\}\right), \tag{9}$$

where Agg aggregates evidence from multiple plausible safe successors. Crucially, $\mathbf{z}_t^{(\text{new})}$ is used only to evaluate the successor embeddings that have already been retrieved based on the preceding context; it is never used to retrieve those contexts themselves. Using the current-step embedding $\mathbf{z}_t^{(\text{new})}$ for retrieval would undermine the intended context-conditioned incompatibility estimation, because whether an action is erroneous or anomalous is often highly dependent on the execution history rather than on the action in isolation. For example, in a web-based task, submitting a form can be entirely appropriate after all required fields have been correctly completed, but the same action may constitute an execution error when performed before the necessary information has been entered. If retrieval were instead performed using the current action itself, the memory could return globally similar but contextually irrelevant safe instances of "submit," causing the erroneous action to appear deceptively normal. The resulting memory score therefore evaluates whether the observed step is appropriate given the execution history that led to it, rather than whether it merely resembles a step that happened to be safe in some other context.

4.3 Failure-Assisted Transition Memory

When a limited annotated support set of failed trajectories $\mathcal{D}_{\text{fail}}$ is available, we aim to leverage such information with minimal additional complexity to further improve detection performance. Specifically, our method introduces an auxiliary failure-matching mechanism that measures whether an observed transition resembles previously identified decisive failure patterns. Importantly, this extension does not modify the safe-dynamics predictor g_θ; instead, failed trajectories are incorporated solely as a non-parametric failure transition memory.

Unlike the safe memory, which stores all step-level embeddings $\mathbf{z}_t^{(i)}, t \in 1, \ldots, T_i$ from each successful trajectory, the failure memory only retains the decisive transition associated with the annotated failure step. For each failed support trajectory, transitions preceding the decisive failure are excluded because they may still correspond to normal execution behavior and could introduce ambiguity into failure-pattern retrieval. Transitions after the decisive failure are also excluded, since our objective is to detect emerging failures as early as possible; the decisive failure transition therefore provides the most informative reference for timely detection. Therefore, the failure memory is constructed as

$$\mathcal{M}_{\text{fail}} = \left\{[\mathbf{z}_0^{(i)}, \mathbf{z}_{t_i^*-1}^{(i)}, \mathbf{z}_{t_i^*}^{(i)}] : \tau_i \in \mathcal{D}_{\text{fail}}, \mathbf{z}_0^{(i)} = \mathbf{z}_q^{(i)}\right\}. \tag{10}$$

where t_i^* denotes the annotated decisive failure step of trajectory τ_i. During inference, analogous to retrieval from the safe transition memory, we use the preceding-step embedding $\mathbf{z}_{t-1}^{(\text{new})}$ as the retrieval query to search the K most relevant failure transitions from $\mathcal{M}_{\text{fail}}$, yielding the $\mathcal{N}_K^{\text{fail}}(t)$.

4.4 An Instantiation of LFS

Safe Dynamics Prediction. Under the safe-only setting, the predictor g_θ is designed to learn the execution dynamics underlying historical successful trajectories. To this end, we formulate its training as a conditional next-step prediction task, where the model predicts the representation of the next execution step given the preceding execution history. Given the sequential nature of execution trajectories, both recurrent neural networks (RNNs) (Elman, 1990) and Transformers (Vaswani et al., 2017) can serve as suitable instantiations of g_θ. Based on empirical comparison, we adopt a long short-term memory (LSTM) network (Hochreiter & Schmidhuber, 1997) as the default architecture.

Specifically, at step t, the LSTM predicts the embedding of the next safe execution step based on the previous-step embedding $\mathbf{z}_{t-1}$ and its hidden state $\mathbf{h}_{t-1}$. We optimize g_θ using a cosine loss,

$$\ell_{\text{dyn}}(t) = 1 - \cos\left(\mathbf{z}_t, g_\theta(\mathbf{h}_{t-1}, \mathbf{z}_{t-1})\right), \tag{11}$$

Minimizing this objective encourages g_θ to capture the transition patterns characteristic of successful executions in the embedding space. Detailed comparisons across different predictor architectures and training objectives are provided in Appendix C.1.

Safe-only Risk Score. During inference under the safe-only setting, we use the immediately preceding step embedding $\mathbf{z}_{t-1}$ as the retrieval context to query the safe transition memory $\mathcal{M}_{\text{safe}}$ and obtain a local safe support set $\mathcal{N}_K^{\text{safe}}(t)$. We then evaluate the compatibility of the observed successor $\mathbf{z}_t$ with this retrieved safe support. Using cosine similarity, the memory-based anomaly score is defined by the most compatible retrieved successor:

$$A_{\text{mem}}(t) = 1 - \max_{(i,j)} \cos(\mathbf{z}_t, \mathbf{z}_{j+1}^{(i)}), \quad (\mathbf{z}_j^{(i)}, \mathbf{z}_{j+1}^{(i)}) \in \mathcal{N}_K^{\text{safe}}(t). \tag{12}$$

A small $A_{\text{mem}}(t)$ indicates that the observed transition is well supported by a historically similar safe transition, whereas a large value indicates greater incompatibility with the retrieved safe support. Since both the dynamics-based score $A_{\text{dyn}}(t)$ and the memory-based score $A_{\text{mem}}(t)$ quantify cosine-based discrepancies in the embedding space, we combine them using a simple parameter-free average:

$$A_{\text{safe}}(t) = \frac{1}{2}\left[A_{\text{dyn}}(t) + A_{\text{mem}}(t)\right]. \tag{13}$$

The $A_{\text{safe}}(t)$ is directly used as the failure-risk score under the safe-only setting, with larger values indicating stronger deviation from the execution patterns observed in successful trajectories.

Failure-Assisted Detection Score. Under the failure-assisted setting, both the safe transition memory $\mathcal{M}_{\text{safe}}$ and the failure transition memory $\mathcal{M}_{\text{fail}}$ are available. Rather than using failed trajectories to retrain the dynamics predictor, we retain the predictor g_{θ^*} learned exclusively from safe trajectories and use the failed transitions only as additional contrastive evidence. This preserves the safe-only dynamics model while allowing the detector to explicitly assess whether a newly observed transition resembles previously observed failure patterns.

For safe transition memory $\mathcal{M}_{\text{safe}}$ and the failure transition memory $\mathcal{M}_{\text{fail}}$, we first use g_{θ^*} to generate their expected successor embedding. Specifically,

$$\begin{cases} \mathcal{G}_{\text{safe}} = \left\{ \hat{\mathbf{z}}_t^{(i)} : \hat{\mathbf{z}}_t^{(i)} = g_{\theta^*}\left(\mathbf{h}_{t-1}^{(i)}, \mathbf{z}_{t-1}^{(i)}\right),\ \mathbf{z}_{t-1}^{(i)} \in [\mathcal{M}_{\text{safe}}]^{\tau_i} \right\}, \\ \mathcal{G}_{\text{fail}} = \left\{ \hat{\mathbf{z}}_{t_i^*}^{(i)} : \hat{\mathbf{z}}_{t_i^*}^{(i)} = g_{\theta^*}\left(\mathbf{h}_{t_i^*-1}^{(i)}, \mathbf{z}_{t_i^*-1}^{(i)}\right),\ \mathbf{z}_{t_i^*-1}^{(i)} \in [\mathcal{M}_{\text{fail}}]^{\tau_i} \right\}. \end{cases} \tag{14}$$

During inference, given a new task $q^{(\text{new})}$, before the embedding $\mathbf{z}_t^{(\text{new})}$ of the current step is observed, we use its preceding-step embedding $\mathbf{z}_{t-1}^{(\text{new})}$ to independently retrieve the $k_1 = \min(K, N_{\text{safe}})$ most relevant safe transitions and the $k_2 = \min(K, N_{\text{fail}})$ most relevant failed transitions, denoted by $\mathcal{N}_{k_1}^{\text{safe}}(t)$ and $\mathcal{N}_{k_2}^{\text{fail}}(t)$, respectively. Meanwhile, the safe dynamics predictor produces

$$\hat{\mathbf{z}}_t^{(\text{new})} = g_{\theta^*}\left(\mathbf{h}_{t-1}^{(\text{new})}, \mathbf{z}_{t-1}^{(\text{new})}\right). \tag{15}$$

Once the embedding $\mathbf{z}_t^{(\text{new})}$ is observed, we represent the current transition and the retrieved historical transitions using the following triplets:

$$\begin{cases} v_{\text{cur}} := \left(\mathbf{z}_{t-1}^{(\text{new})}, \mathbf{z}_t^{(\text{new})}, \hat{\mathbf{z}}_t^{(\text{new})}\right), \\ \mathcal{V}_{\text{safe}} := \left\{ \left(\mathbf{z}_{j-1}^{(\text{safe})}, \mathbf{z}_j^{(\text{safe})}, \hat{\mathbf{z}}_j^{(\text{safe})}\right) : \left(\mathbf{z}_{j-1}^{(\text{safe})}, \mathbf{z}_j^{(\text{safe})}\right) \in \mathcal{N}_{k_1}^{\text{safe}}(t) \right\}, \\ \mathcal{V}_{\text{fail}} := \left\{ \left(\mathbf{z}_{j^*-1}^{(\text{fail})}, \mathbf{z}_{j^*}^{(\text{fail})}, \hat{\mathbf{z}}_{j^*}^{(\text{fail})}\right) : \left(\mathbf{z}_{j^*-1}^{(\text{fail})}, \mathbf{z}_{j^*}^{(\text{fail})}\right) \in \mathcal{N}_{k_2}^{\text{fail}}(t) \right\}. \end{cases} \tag{16}$$

Accordingly, $|\mathcal{V}_{\text{safe}}| = k_1$ and $|\mathcal{V}_{\text{fail}}| = k_2$. Each triplet jointly characterizes the observed transition and its consistency with the dynamics learned from successful executions.

To compare these triplets from complementary perspectives, we define $\text{norm}(\mathbf{z}) = \frac{\mathbf{z}}{\max(\|\mathbf{z}\|_2, \epsilon)}$ with $\epsilon = 10^{-6}$, and construct three representations:

$$\mathbf{s}_t = \text{norm}(\mathbf{z}_t), \qquad \mathbf{d}_t = \text{norm}\left(\mathbf{z}_t - \mathbf{z}_{t-1}\right), \qquad \mathbf{e}_t = \text{norm}\left(\bar{\mathbf{z}}_t - \bar{\hat{\mathbf{z}}}_t\right), \tag{17}$$

where $\bar{\mathbf{z}}_t = \mathrm{norm}(\mathbf{z}_t)$ and $\bar{\hat{\mathbf{z}}}_t = \mathrm{norm}(\hat{\mathbf{z}}_t)$. The $\mathbf{s}_t$ characterizes the semantic state of the current execution step in the embedding space; The $\mathbf{d}_t$ describes the local transition direction from the preceding step to the current step; and the $\mathbf{e}_t$ characterizes the prediction-residual pattern between the observed successor and the successor expected by the safe dynamics model. *These three representations therefore capture complementary information about **where** the current step lies, **how** the execution evolves, and **how** the observed evolution deviates from the learned safe dynamics.*

The same transformation is applied to v_{cur} and every element in $\mathcal{V}_{\mathrm{safe}}$ and $\mathcal{V}_{\mathrm{fail}}$. For each retrieved safe candidate j, we define its compatibility with the current transition as

$$\kappa_j^{\mathrm{safe}}(t) = \frac{1}{3}\left[\cos(\mathbf{s}_t, \mathbf{s}_j) + \cos(\mathbf{d}_t, \mathbf{d}_j) + \cos(\mathbf{e}_t, \mathbf{e}_j)\right]. \tag{18}$$

Similarly, for each retrieved failed candidate j^*, we compute

$$\kappa_{j^*}^{\mathrm{fail}}(t) = \frac{1}{3}\left[\cos(\mathbf{s}_t, \mathbf{s}_{j^*}) + \cos(\mathbf{d}_t, \mathbf{d}_{j^*}) + \cos(\mathbf{e}_t, \mathbf{e}_{j^*})\right]. \tag{19}$$

A larger $\kappa_j^{\mathrm{safe}}(t)$ indicates stronger compatibility with a historical successful transition, whereas a larger $\kappa_{j^*}^{\mathrm{fail}}(t)$ indicates stronger resemblance to a previously observed failed transition. Because not every retrieved transition is equally informative, we retain only the most compatible candidates from each memory. Let

$$c_1^{\mathrm{safe}}(t) \geq \cdots \geq c_{k_1}^{\mathrm{safe}}(t), \qquad c_1^{\mathrm{fail}}(t) \geq \cdots \geq c_{k_2}^{\mathrm{fail}}(t) \tag{20}$$

denote the corresponding compatibility scores sorted in descending order. We then average the top r scores, where $r \leq \min(k_1, k_2)$:

$$\bar{\kappa}^{\mathrm{safe}}(t) = \frac{1}{r}\sum_{i=1}^{r} c_i^{\mathrm{safe}}(t), \qquad \bar{\kappa}^{\mathrm{fail}}(t) = \frac{1}{r}\sum_{i=1}^{r} c_i^{\mathrm{fail}}(t). \tag{21}$$

Using the top-r average instead of a single nearest neighbor reduces sensitivity to individual retrieved examples while still emphasizing the most relevant local evidence.

Finally, we define the failure-assisted contrast score as

$$A_{\mathrm{assist}}(t) = \bar{\kappa}^{\mathrm{fail}}(t) - \left(\bar{\kappa}^{\mathrm{safe}}(t) + \delta_t\right), \tag{22}$$

where δ_t is an adaptive margin that reflects the local dispersion of the retrieved safe transitions.

$$\delta_t = \min(\bar{\kappa}^{\mathrm{safe}}(t) - \underbrace{c_r^{\mathrm{safe}}(t)}_{\text{top-}r}, \underbrace{c_1^{\mathrm{safe}}(t)}_{\text{top-1}} - \bar{\kappa}^{\mathrm{safe}}(t)), \tag{23}$$

The δ_t measures the distance between the average safe compatibility and the closer boundary of the retrieved top-r safe compatibility interval $[c_r^{\mathrm{safe}}(t), c_1^{\mathrm{safe}}(t)]$. A larger δ_t indicates that the retrieved safe transitions provide more diverse compatibility evidence, and therefore stronger failure evidence is required to distinguish a potential failure from ambiguous safe behaviors. Conversely, when the retrieved safe transitions are highly consistent, δ_t becomes smaller, allowing the detector to respond to weaker failure evidence. Therefore, the adaptive margin reduces false alarms caused by uncertain safe contexts while maintaining sensitivity to transitions that are more compatible with historical failure patterns. Details of the online alarm are provided in Appendix B.5.

5 Experiments

We evaluate LFS on two public agent-trajectory benchmarks, AFTraj-2K (Zhang et al., 2026a) and AgentProcessBench (Fan et al., 2026). For the safe-only setting, both model training and threshold calibration use successful trajectories exclusively. For the failure-assisted setting, failed trajectories are introduced only as retrieval memory and never used to retrain the safe-dynamics predictor for our method. Detailed dataset statistics, implementation details, and evaluation protocols are provided in Appendix B. Our experiments are designed to answer three questions:

- **[Q1]** Under the safe-only setting, how does LFS compare with existing baselines in both detection effectiveness and online inference efficiency?
- **[Q2]** When limited failed trajectories become available, how effectively can LFS exploit them to improve early failure detection?
- **[Q3]** With substantially less failure supervision, how competitive is failure-assisted LFS compared with fully supervised early failure detectors?

5.1 SAFE-ONLY EARLY FAILURE DETECTION

To answer **[Q1]**, we evaluate LFS and the baseline methods under the safe-only setting. For a fair comparison, we use *Qwen2.5-3B-Instruct* (Yang et al., 2024) as the fixed encoder $E(\cdot)$ for all methods. The online detection results are reported in Table 1. We have the following observations:

Table 1: Safe-only performance on the AgentProcessBench and AFTraj-2K. The **top-1** and **top-2** values are highlighted. The "N/A" in the ASS column indicates that no true failure trajectories were detected by the corresponding method.

Method	AUROC (%) ↑	AUPRC (%) ↑	Exact-F1 (%) ↑	Step-Acc (%) ↑	ASS ↓
AgentProcessBench					
Qwen2.5-3B (Prompt)	52.32	19.59	2.53	1.67	6.844
Qwen2.5-3B (ICL)	53.45	19.78	**5.85**	**4.17**	**6.250**
ATS	**58.74**±0.29	**26.23**±0.41	4.47±1.36	3.00±0.95	12.665 ± 0.667
OAT	44.83±2.67	16.96±0.89	1.78±1.46	1.17 ± 0.95	16.418 ± 2.192
Safe-FSM	53.43±0.83	23.94±1.68	1.67±1.84	1.00 ± 1.09	**4.002**±1.707
LFS (Qwen2.5-3B)	**73.32**±0.59	**36.36**±0.65	**12.41**±1.44	**7.17**±0.95	7.141±3.771
Coding from AFTraj-2K					
Qwen2.5-3B (Prompt)	69.65	53.62	36.67	29.73	**0.522**
Qwen2.5-3B (ICL)	74.57	49.44	50.00	43.24	**0.522**
ATS	85.56±0.64	68.76±1.16	57.74±0.34	59.46±0.00	0.691±0.031
OAT	**92.24**±0.22	**72.56**±0.43	**67.06**±4.73	**64.32**±1.21	0.919±0.291
Safe-FSM	81.60±0.58	53.77±4.91	28.64±16.92	23.24±14.25	1.520±0.831
LFS (Qwen2.5-3B)	**96.66**±0.05	**79.15**±1.17	**60.76**±3.28	**64.32**±2.26	0.566 ± 0.182
Math from AFTraj-2K					
Qwen2.5-3B (Prompt)	50.00	10.62	6.25	3.45	7.200
Qwen2.5-3B (ICL)	59.48	8.76	0.00	0.00	N/A
ATS	75.13±0.60	11.02±0.36	1.87±1.70	1.03±0.94	9.367±1.057
OAT	88.77±0.85	**59.38**±11.04	**41.16**±8.32	**36.90**±8.93	2.671±0.300
Safe-FSM	**92.05**±0.94	50.45±9.77	10.68±7.50	8.97±6.72	**1.961**±0.748
LFS (Qwen2.5-3B)	**94.91**±0.12	**76.64**±0.32	**67.88**±1.22	**63.79**±2.11	**0.979**±0.075
Agentic from AFTraj-2K					
Qwen2.5-3B (Prompt)	55.94	19.10	5.00	2.94	4.200
Qwen2.5-3B (ICL)	**72.86**	**39.59**	**20.56**	**16.18**	**2.062**
ATS	35.65±0.95	12.83±0.61	5.90±0.11	4.41±0.00	3.745±0.130
OAT	68.43±1.54	27.51±2.21	11.85±0.61	7.35±0.00	**2.170**±0.906
Safe-FSM	56.54±1.08	15.38±0.46	0.00±0.00	0.00±0.00	5.435±0.706
LFS (Qwen2.5-3B)	**85.14**±0.28	**48.86**±1.10	**14.45**±1.67	**9.71**±1.32	2.242±0.305
Average on AFTraj-2K					
Qwen2.5-3B (Prompt)	56.78	14.77	14.71	9.20	2.368
Qwen2.5-3B (ICL)	68.15	18.50	23.58	16.56	**1.418**
ATS	64.00±0.90	10.60±0.26	21.14±0.53	15.71±0.34	2.189±0.050
OAT	**84.64**±0.63	**49.16**±5.66	**38.74**±3.62	**30.80**±3.11	1.976±0.237
Safe-FSM	78.85±0.92	25.24±1.58	11.22±5.58	8.47±4.64	2.657±0.246
LFS (Qwen2.5-3B)	**93.15**±0.11	**68.36**±0.31	**48.38**±1.13	**41.35**±1.27	**1.073**±0.104

- Compared with the training-free methods based on direct prompting and in-context learning, LFS achieves substantially stronger ranking quality (AUROC and AUPRC) and exact failure localization (Exact-F1) in almost all cases across the two benchmarks. On AFTraj-2K, LFS also achieves the best average temporal localization and step-level accuracy, as reflected by ASS and Step-Acc. On AgentProcessBench, its improvements are mainly observed in ranking quality and Exact-F1 **[Q1]**.
- Compared with training-required safe-only monitors, including ATS, OAT, and Safe-FSM, LFS shows the strongest overall performance on AFTraj-2K, with consistent advantages in ranking quality and failure localization across the three domains. On AgentProcessBench, LFS also achieves the best AUROC, AUPRC, and Exact-F1, but does not dominate all localization-related metrics: existing sequential monitors can achieve better ASS and Step-Acc **[Q1]**.

Figure 2 further evaluates online inference efficiency. Unlike prompt-based or in-context LLM auditors that repeatedly invoke autoregressive generation, LFS encodes each newly observed step once and performs subsequent monitoring with lightweight prediction, retrieval, and scoring operations. Consequently, the monitoring overhead is small relative to representation extraction and substantially lower than generative LLM-based auditing, making LFS suitable for continuous online monitoring **[Q1]**.

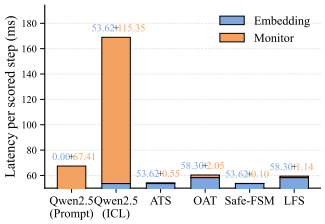

Figure 2: Average online inference latency per step.

5.2 FAILURE-ASSISTED EARLY FAILURE DETECTION

Following AgentForesight (Zhang et al., 2026a), we use the same test split for direct comparison. Our safe-dynamics predictor is trained only on the safe trajectories from the training set, while subsets

of the failed training trajectories are used exclusively to construct the failure transition memory, without further model training or fine-tuning. To answer **[Q2]** and **[Q3]**, we progressively vary the proportion of failed trajectories used by LFS from 5% to 50%, where 100% corresponds to the full failed training set available to the fully supervised methods. For comparison, we also report the results of AgentForesight and its supervised baselines from the original evaluation. The partial results are summarized in Table 2, while the complete comparison is provided in Appendix C.4.

Table 2: Failure-Assisted performance compared with the fully supervised methods on the AFTraj-2K. The **top-1** and **top-2** values are highlighted.

Method	Usage of Failed Support	Math		Coding		Agentic		Overall	
		Exact-F1(%) ↑	ASS ↓	Exact-F1(%) ↑	ASS ↓	Exact-F1(%) ↑	ASS ↓	Exact-F1(%) ↑	ASS ↓
Safe-only LFS (Qwen2.5-3B-Instruct)	0%	67.88	0.979	60.76	**0.566**	14.45	2.242	48.38	1.073
Qwen2.5-7B-Instruct-based methods									
Perplexity-7B	100%	2.31	4.09	26.56	2.19	16.57	2.50	14.11	3.02
ToT-7B	100%	20.38	4.40	7.02	4.06	24.84	2.13	18.52	3.39
Reflexion-7B	100%	16.57	4.39	9.52	4.50	**39.13**	1.44	23.38	3.17
AgentDebug-7B	100%	0.00	–	28.57	4.05	2.82	**1.00**	9.63	3.76
AgentForesight-7B	100%	77.36	0.96	78.87	**0.18**	**48.70**	**0.54**	**66.44**	**0.59**
Failed trajectories used as contrastive transition memory									
Failure-Assisted LFS (Qwen2.5-3B-Instruct)	5%	75.11	0.721	68.49	1.200	10.63	2.618	53.15	1.143
	10%	83.78	0.368	69.29	1.183	11.44	2.421	57.10	0.964
	15%	85.37	0.323	70.60	1.149	12.08	2.305	58.26	0.921
	20%	85.87	**0.310**	74.26	1.058	12.82	2.247	59.73	0.875
	25%	85.72	0.325	74.59	1.050	12.85	2.202	59.81	0.874
	30%	85.87	0.318	76.26	1.012	13.50	2.141	60.43	0.860
	35%	85.87	**0.290**	78.08	0.967	14.33	2.136	61.11	0.838
	40%	**86.02**	0.335	**80.32**	0.924	16.43	1.999	**62.15**	**0.835**
	45%	**86.02**	0.376	**79.89**	0.924	14.26	2.100	61.39	0.870
	50%	85.72	0.395	79.83	0.929	15.97	2.029	61.76	0.869

- **Limited failure supervision provides clear complementary evidence.** Compared with safe-only LFS, incorporating failed transitions generally improves the overall Exact-F1 and reduces ASS as more failure support becomes available. This indicates that a limited number of annotated failure transitions can provide useful complementary signals and also demonstrates that our failure-assisted design can effectively exploit limited failure supervision. The gains gradually saturate as more failed trajectories are added, suggesting that a moderate-sized failure memory already captures a substantial portion of reusable failure patterns. This further supports the effectiveness of LFS in scenarios where annotated unsafe trajectories are limited **[Q2]**.
- **The benefit of failure memory is domain dependent.** While Math and Coding benefit substantially from additional failed trajectories, the improvement on Agentic tasks is smaller and less monotonic. This suggests that retrieval-based failure assistance is most effective when failure transitions exhibit recurring and locally comparable patterns, whereas more heterogeneous execution paths and failure modes make the direct reuse of historical failures more difficult **[Q2]**.
- **LFS remains competitive with fully supervised detectors using substantially less failure supervision.** With only a fraction of the failed training trajectories stored as transition memory, LFS substantially outperforms most supervised LLM-based baselines and approaches the overall performance of AgentForesight-7B. On Math and Coding, LFS can even surpass AgentForesight in Exact-F1 with partial failure support, while AgentForesight remains stronger overall, particularly on Agentic tasks **[Q3]**.

5.3 More Experimental Results

The effects of the predictor architecture and training objective are studied in Appendix C.1; the failure-memory representation and top-r aggregation are analyzed in Appendix C.2; and mixed-domain versus domain-specific training is examined in Appendix C.3.

6 Conclusion

In this work, we study *safe-only early failure detection* for LLM-based agent systems and propose LFS, which combines a learned safe-dynamics predictor with a non-parametric safe transition memory to capture complementary global temporal regularities and local contextual consistency. When limited failed trajectories are available, LFS further incorporates their decisive transitions as contrastive failure memory without retraining or fine-tuning the safe monitor. Experiments on two benchmarks show that LFS provides strong failure discrimination and localization under the safe-only setting while introducing only lightweight online monitoring overhead. With limited failure support, performance can be further improved, while remaining competitive with fully supervised detectors using substantially less failure supervision.

AI USE STATEMENT

Large language model (LLM) tools were used only to assist with language polishing and improving the clarity of the manuscript. All research ideas, methodology, experiments, analysis, and conclusions were developed and verified by the authors.

REFERENCES

Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation, 2025. URL https://arxiv.org/abs/2503.11926.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei A. Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-4082. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/b1041e52d3be19f0a9bc491657488e4a-Abstract-Datasets_and_Benchmarks_Track.html. Datasets and Benchmarks Track.

Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. TRAIL: Trace reasoning and agentic issue localization, 2025. URL https://arxiv.org/abs/2505.08638.

Sunny Dubey. Real-time detection and repair of LLM agent failures, 2026. URL https://arxiv.org/abs/2608.02464.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Shengda Fan, Xuyan Ye, Yupeng Huo, Zhi-Yuan Chen, Yiju Guo, Shenzhi Yang, Wenkai Yang, Shuqi Ye, Jingwen Chen, Haotian Chen, et al. Agentprocessbench: Diagnosing step-level process quality in tool-using agents. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 8823–8834, 2026.

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

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

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

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Neil Kale, Chen Bo Calvin Zhang, Kevin Zhu, Ankit Aich, Paula Rodriguez, Scale Red Team, Christina Q. Knight, and Zifan Wang. Reliable weak-to-strong monitoring of LLM agents, 2025. URL https://arxiv.org/abs/2508.19461.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/4a5876b450b45371f6cfe5047ac8cd45-Abstract.html.

Jonathan Kutasov, Yuqi Sun, Paul Colognese, Teun van der Weij, Linda Petrini, Chen Bo Calvin Zhang, John Hughes, Xiang Deng, Henry Sleight, Tyler Tracy, Buck Shlegeris, and Joe Benton. SHADE-Arena: Evaluating sabotage and monitoring in LLM agents, 2025. URL https://arxiv.org/abs/2506.15740.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for Mind exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36, pp. 51991–52008, 2023. doi: 10.52202/075280-2264. URL https://proceedings.neurips.cc/paper/2023/hash/a3621ee907def47c1b952ade25c67698-Abstract-Conference.html.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023.

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

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/25ae35b5b1738d80f1f03a8713e405ec-Abstract-Conference.html.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Aymeric Roucher, A Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. smolagents: A smol library to build great agentic systems. *Hugging Face*, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent laboratory: Using LLM agents as research assistants. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 5977–6043. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.320. URL https://aclanthology.org/2025.findings-emnlp.320/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a4b6ad6b48850c0c331d1259fc66a69c-Abstract-Conference.html.

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

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher 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*, pp. 2369–2380, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Samuel Yeh, Yiwen Zhu, Shaleen Deep, and Sharon Li. Tracing agentic failure from the flow of success, 2026. URL https://arxiv.org/abs/2607.12747.

Boxuan Zhang, Jianing Zhu, Zeru Shi, Dongfang Liu, and Ruixiang Tang. AgentForesight: Online auditing for early failure prediction in multi-agent systems, 2026a. URL https://arxiv.org/abs/2605.08715.

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

Heng Zhang, Yuling Shi, Xiaodong Gu, Haochen You, Zijian Zhang, Lubin Gan, Yilei Yuan, and Jin Huang. GraphTracer: Graph-guided failure tracing in LLM agents for robust multi-turn deep search, 2025a. URL https://arxiv.org/abs/2510.10581.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76583–76599. PMLR, 2025b. URL https://proceedings.mlr.press/v267/zhang25cq.html.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. ProcessBench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 1009–1024. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.50. URL https://aclanthology.org/2025.acl-long.50/.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/4410c0711e9154a7a2d26f9b3816d1ef-Abstract-Conference.html.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. Where LLM agents fail and how they can learn from failures, 2025. URL https://arxiv.org/abs/2509.25370.

Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 80569–80611. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhuge25a.html.

APPENDIX CONTENTS

A Related Work **14**

A.1 Failure Analysis and Post-hoc Attribution 14

A.2 Online Agent Monitoring and Early Failure Detection 14

B Experimental Setups **15**

B.1 Details of Datasets 15

B.2 Details of Baselines 15

B.3 Details of Evaluation Metrics 16

B.4 Details of Implementation 17

B.5 Details of Online Alarm 18

C Additional Experimental Results **19**

C.1 Selection of g_θ and ℓ_{dyn} 19

C.2 Failure-Memory Representation and Aggregation 19

C.3 Domain Mixing and Domain Specific Training 20

C.4 Overall Comparison between Failure-Assisted LFS and Fully Supervised Baselines 20

A Related Work

A.1 Failure Analysis and Post-hoc Attribution

As agentic executions become longer and involve more interacting components, recent work has increasingly focused on understanding why an execution fails rather than merely measuring its final success. MAST (Cemri et al., 2025) develops a taxonomy of multi-agent failures covering system design, inter-agent misalignment, and task verification, while TRAIL (Deshpande et al., 2025) and AgentDebug (Zhu et al., 2025) study complementary categories of reasoning, planning, memory, action, and system-level errors. These studies provide useful abstractions and annotated trajectories for characterizing failure behaviors.

Beyond failure categorization, a growing body of work studies *failure attribution*, which aims to localize the responsible agent or decisive error step after an execution has failed. Who&When (Zhang et al., 2025b) formulates automated attribution as identifying the responsible agent and failure step from a completed trajectory. AgenTracer (Zhang et al., 2026b) learns a specialized tracer through counterfactual replay and programmed fault injection, while GraphTracer (Zhang et al., 2025a) explicitly models cross-agent information dependencies for failure localization. DoVer (Ma et al., 2026) further evaluates whether an attributed failure can be validated through intervention, rather than relying solely on textual diagnosis. Related reflection-based approaches use execution outcomes or feedback to improve subsequent attempts (Shinn et al., 2023).

These approaches provide valuable retrospective evidence for debugging and recovery, but their primary decision is generally made after the complete trajectory or task outcome is available. In contrast, our goal is *online early failure detection*: after each newly observed execution step, the monitor must decide whether to continue execution or raise an alarm using only the current trajectory prefix. This causal restriction distinguishes our setting from post-hoc attribution because future actions and the eventual outcome are unavailable when the decision is made.

A.2 Online Agent Monitoring and Early Failure Detection

Runtime monitoring has been studied as a mechanism for identifying undesirable or unreliable agent behavior during execution. Chain-of-thought monitoring investigates whether intermediate reasoning can reveal behaviors such as reward hacking, while also showing that optimization against a monitor may alter or obscure the monitored reasoning process (Baker et al., 2025). SHADE-Arena evaluates sabotage and monitoring in tool-using agents, and weak-to-strong monitoring studies how monitoring scaffolds affect oversight when the monitored model is stronger than the monitor (Kutasov et al., 2025; Kale et al., 2025). These works mainly address malicious, deceptive, or misaligned behavior rather than ordinary task-execution failures.

A related line of work evaluates intermediate reasoning or execution traces. ProcessBench (Zheng et al., 2025) provides step-level supervision for localizing errors in mathematical reasoning, while Agent-as-a-Judge (Zhuge et al., 2025) evaluates agent trajectories beyond final-answer correctness. However, such evaluators are not necessarily designed for a strict causal first-alarm protocol in which the monitor must make an irrevocable online decision after every newly observed step.

More directly related to our setting, AgentForesight (Zhang et al., 2026a) formulates online auditing of multi-agent execution, where an auditor observes only the current trajectory prefix and attempts to identify the earliest decisive error. Its AgentForesight-7B model is trained using successful and failed trajectories together with decisive-error supervision. AgentTrajectorySentinel (ATS) (Dubey, 2026) instead trains a lightweight one-class sequential monitor from healthy execution telemetry and combines learned monitoring with deterministic verification. Our work adopts the same causal requirement of making decisions from the observed prefix, but focuses on learning the conditional evolution of successful semantic execution trajectories. OAT (Yeh et al., 2026) models the latent flow of successful trajectories using neural controlled differential equations (Kidger et al., 2020). OAT uses successful executions to construct an unsupervised attribution model and identifies suspicious steps within a failed trajectory. LFS differs in both problem formulation and modeling. First, LFS operates under a strictly online first-alarm protocol and calibrates decisions using successful trajectories only. Second, rather than relying on a single learned trajectory model, LFS combines two complementary estimates of successful execution: a parametric next-step predictor that captures global safe execution dynamics and a non-parametric transition memory that retrieves actual successors following similar

historical contexts. The resulting detector evaluates whether each newly observed step remains compatible with the execution history that precedes it.

B Experimental Setups

B.1 Details of Datasets

Following prior work (Zhang et al., 2026a; Yeh et al., 2026), we use two recent public agent-trajectory benchmarks, AFTraj-2K (Zhang et al., 2026a) and AgentProcessBench (Fan et al., 2026) to evaluate our method and all baselines. The detailed statistics are provided in the following tables.

AFTraj-2K. We follow the official AgentForesight test partition for fair and direct comparison. It contains 332 trajectories (169 safe and 163 failed) from Coding, Math, and Agentic tasks. The remaining safe trajectories are separated into model training (85%) and validation (15%) sets. All failed trajectories outside the test partition are excluded from the safe-only experiments. For failure-assisted LFS, we additionally form an optional support pool after removing trajectories whose normalized task group overlaps the safe validation or test groups. Table 3 reports the exact statistics of AFTraj-2K and our experimental splitting.

Table 3: The statistics of AFTraj-2K and specific experiments protocol.

Domains	**Coding**	**Math**	**Agentic**	**Overall**
Task Source	HumanEval+, MBPP+ (Liu et al., 2023)	MATH-500 (Hendrycks et al., 2021)	GAIA (Mialon et al., 2024), HotpotQA (Yang et al., 2018)	—
(Multi-Agent Systems)	AutoGen (Wu et al., 2024), MetaGPT (Hong et al., 2024)	AutoGen (Wu et al., 2024)	Smolagents (Roucher et al., 2025)	—
Safe	361	395	402	1,158
Unsafe	247	397	470	1,114
Total	608	792	872	2,272
Avg. # turns	10.0	16.2	8.4	11.5
Train Set (Safe-only)	261	287	307	855
Validation Set (Safe-only)	46	51	41	138
Test Set (Safe)	54	58	57	169
Test Set(Unsafe)	37	58	68	163
Optional Support Set (Unsafe)	182	299	332	813

AgentProcessBench (APB). APB contains 1,000 process-annotated trajectories from 200 query groups and four equally sized sources: BFCL (Patil et al., 2025), GAIA (Mialon et al., 2024), HotpotQA (Yang et al., 2018), and Tau2 (Barres et al., 2025). A trajectory is safe iff none of its step labels is -1; for a failed trajectory, t^* is the first step labeled -1. We reserve one trajectory per query group as a fixed test set and split the remaining safe trajectories $85/15$ into training and validation sets.

Table 4: The AgentProcessBench protocol in the safe-only experiments.

Task Source	Train Set (Safe-only)	Validation Set (Safe-only)	Test Set (Safe-only)	Test Set (Unsafe-only)
BFCL (Patil et al., 2025)	44	8	14	36
GAIA (Mialon et al., 2024)	45	8	14	36
HotpotQA (Yang et al., 2018)	99	17	30	20
Tau2 (Barres et al., 2025)	72	13	22	28
Total	260	46	80	120

Across the released data, AFTraj-2K trajectories have "3/11/74" corresponding to "minimum/median/maximum" steps, while APB trajectories have "3/17/205". The longer and more heterogeneous APB executions make its exact first-error localization substantially more difficult.

Figure 3 shows the normalized position of the first decisive error across failed trajectories. Errors occur throughout both benchmarks rather than at a fixed execution stage: AFTraj-2K has a mean/median position of 0.47/0.43, while AgentProcessBench is slightly later at 0.51/0.53. This broad distribution motivates online monitoring throughout the complete execution process.

B.2 Details of Baselines

We compare methods under the same prefix-only input, fixed splits, and safe-validation calibration. Table 5 states the role of each baseline and the adaptation needed for strict online evaluation.

For completeness, the AgentForesight results in Table 2 have different supervision budgets. The generic open- and closed-source LLM auditors and AgentDebug are prompted directly and are not

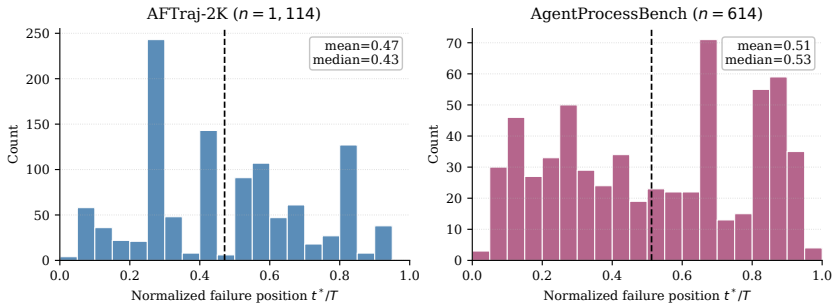

Figure 3: Distribution of the decisive-error step (normalized by trajectory length T).

Table 5: Safe-only baselines. "Adapted" indicates that the published idea is retained but its inputs or scoring are modified to satisfy the common embedding and online-causality protocol.

Method	Safe information used	Online score	Purpose / protocol note
Qwen2.5 Prompt	None	Normalized next-token probability of `ERROR` versus `SAFE`	Frozen Qwen2.5-3B-Instruct judges each visible prefix; no fitting or demonstrations.
Qwen2.5 ICL	Three retrieved safe transitions	Same binary next-token score as Prompt	Demonstrations are predecessor–successor transitions retrieved from safe training memory using the preceding step only.
ATS	Safe embedding sequences	Echo-state ensemble disagreement with CUSUM accumulation	Embedding-only adaptation because AFTraj/APB do not expose the complete runtime telemetry used by ATS.
OAT	Safe continuous trajectories	Squared error from the learned success flow	Adapted to produce its expectation before consuming the current step, preventing future/current-input leakage.
Safe-FSM	Safe role/action/state transitions	Smoothed negative log transition likelihood	Adapted safe-only first-order Markov monitor over discretized semantic states and observed role/action transitions.
LFS (Safe dynamics)	Safe embedding sequences	Next-step prediction discrepancy	Parametric component of LFS; isolates learned history-dependent dynamics.
LFS (Transition memory)	Safe predecessor-successor pairs	Cosine distance to the best successor following a similar safe context	Non-parametric component of LFS; isolates contextual historical support.

trained on AFTraj. Perplexity-7B and ToT-7B tune a scalar decision threshold on a labeled held-out training split; Reflexion-7B tunes its threshold on labeled held-out trajectories. AgentForesight-7B is fully supervised: Stage 1 uses 1,902 failure-boundary pairs derived from 951 non-test failed trajectories, and Stage 2 uses all 1,940 non-test trajectories (989 safe and 951 failed) (Zhang et al., 2026a). In contrast, the LFS rows use failed examples only as frozen decisive-transition memory and retain a predictor trained exclusively on safe trajectories.

B.3 Details of Evaluation Metrics

Following prior work (Zhang et al., 2026a; Yeh et al., 2026), we evaluate early failure detection using five metrics. AUROC and AUPRC measure threshold-independent ranking quality, Exact-F1 and Step-Acc measure exact localization of the decisive failure step, and ASS measures the temporal deviation between the predicted alarm step and the annotated decisive failure step. For AFTraj-2K, each trajectory contains exactly one decisive error step. For AgentProcessBench, t_i^* is defined as the first step annotated with label -1, rather than a later erroneous step. For each test trajectory i, let $t_{\text{alarm},i}$ denote the first alarm step produced by the monitor, with $t_{\text{alarm},i} = -1$ if no alarm is raised. For an unsafe trajectory, let t_i^* denote its annotated decisive failure step. Let $\mathcal{U}$ denote the set of unsafe test trajectories, and define

$$\mathcal{D} = \{i \in \mathcal{U} : t_{\text{alarm},i} \neq -1\}, \qquad \mathcal{C} = \{i \in \mathcal{U} : t_{\text{alarm},i} = t_i^*\}, \tag{24}$$

where $\mathcal{D}$ contains the detected unsafe trajectories and $\mathcal{C}$ contains those whose decisive failure step is exactly localized.

Exact-F1 and Step-Acc. We first define the precision and recall of exact failure localization as

$$P_{\text{exact}} = \frac{|\mathcal{C}|}{|\mathcal{D}|}, \qquad R_{\text{exact}} = \frac{|\mathcal{C}|}{|\mathcal{U}|}. \tag{25}$$

Exact-F1 is then

$$\text{Exact-F1} = \frac{2P_{\text{exact}}R_{\text{exact}}}{P_{\text{exact}} + R_{\text{exact}}}, \tag{26}$$

while Step-Acc is

$$\text{Step-Acc} = \frac{|\mathcal{C}|}{|\mathcal{U}|}. \tag{27}$$

Thus, Step-Acc measures the proportion of unsafe trajectories whose decisive failure step is exactly identified, whereas Exact-F1 additionally penalizes detected failures whose alarm occurs at an incorrect step.

ASS. For detected unsafe trajectories, ASS measures the average absolute distance between the predicted alarm step and the annotated decisive failure step:

$$\text{ASS} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \left| t_{\text{alarm},i} - t_i^* \right|. \tag{28}$$

A lower ASS indicates that the alarm is temporally closer to the decisive failure step. ASS is defined only over detected unsafe trajectories and is reported as N/A when $|\mathcal{D}| = 0$.

AUROC and AUPRC. AUROC and AUPRC evaluate the ranking quality of the continuous detection score independently of the alarm threshold. For each unsafe test trajectory, the score at its annotated decisive failure step t_i^* is treated as a positive sample. The negative samples consist of all scored steps from safe test trajectories. Non-decisive steps from unsafe trajectories are excluded because their correctness is not fully annotated. Accordingly, these metrics measure whether decisive failure steps receive higher risk scores than steps observed during successful executions.

B.4 Details of Implementation

Execution representation. The encoder is frozen for every embedding-based method. The task query and each observable step are serialized separately, encoded once, and cached; no future step is included in the serialization. Table 6 specifies the representation shared by LFS and compatible baselines.

Table 6: Frozen execution-encoder configuration.

Item	Setting
Backbone	Qwen2.5-3B-Instruct
Layer	Final transformer hidden state
Pooling	Masked mean over valid tokens
Embedding dimension	2,048
Normalization	None (raw hidden representation)
Encoder precision	BF16, evaluation mode, no gradients
Maximum input length	2,048 tokens

Safe-dynamics predictor. The default predictor is a task-initialized LSTM with a 2,048-dimensional hidden state and a two-layer $2048 \to 4096 \to 2048$ GELU prediction head. The query embedding initializes the recurrent execution; at step t, only embeddings from steps before t are consumed before $\hat{\mathbf{z}}_t$ is emitted. The observed $\mathbf{z}_t$ is used only to compute its cosine discrepancy and is incorporated into the recurrent state after scoring. Table 7 lists the fixed training settings.

The training loss is first averaged over steps within a trajectory and then over trajectories so that long executions do not dominate. Checkpoint selection uses safe validation loss only (source-balanced for APB); no failed trajectory participates. The transition memory is built only from safe training trajectories. Deterministic retrieval and frozen-Qwen baselines are run once, whereas stochastic neural methods report the mean and standard deviation over five seeds. Failure-assisted results additionally average over ten independently sampled, domain-stratified failure-memory draws.

Table 7: Default LFS training and retrieval configuration.

Item	Setting
Predictor parameters	58.75M
Training objective / score	$1 - \cos(\hat{\mathbf{z}}_t, \mathbf{z}_t)$
Optimizer	AdamW
Learning rate / weight decay	10^{-4} / 10^{-4}
Batch size / maximum epochs	256 / 100
Gradient clipping	1.0
Checkpoint criterion	Minimum mean safe-validation trajectory loss
Safe-memory retrieval	Cosine similarity, $K = 100$
Safe successor aggregation	Best matching successor (top-1)
Failure-assisted aggregation	$r = 5$ compatible transitions
Neural seeds	$\{42, 43, 44, 45, 46\}$

For the adapted baselines, ATS uses a 64-dimensional safe-PCA projection and an ensemble of eight 128-state echo-state networks (spectral radius 0.9, leak 0.3, ridge 0.01). OAT uses a 64-dimensional projection with 64 hidden and vector-field dimensions, AdamW with learning rate 4×10^{-5}, and a maximum of 300 epochs. Safe-FSM uses 32 k-means semantic states and additive smoothing of 1.0. These values are fixed before test evaluation.

Hardware and software environment. Table 8 reports the environment used for the local experiments. Each model, seed, or support-memory replicate runs on a single GPU; independent jobs may be scheduled concurrently on different GPUs, but we do not use distributed training.

Table 8: Hardware and software environment.

Item	Setting
GPU per run	NVIDIA GeForce RTX 4090 (24 GB)
Operating system	Linux
Python	3.11.0
PyTorch	2.8.0+cu126
CUDA build	12.6
Transformers	5.2.0
Training mode	Single GPU; no distributed training

B.5 Details of Online Alarm

For both the safe-only detection score $A_{\text{safe}}(t)$ and the failure-assisted detection score $A_{\text{assist}}(t)$, we calibrate the alarm threshold using only successful validation trajectories. Specifically, for each safe validation trajectory $\tau_i \in \mathcal{D}_{\text{val}}^{\text{safe}}$, we first compute its maximum detection score over the entire execution:

$$m_i = \max_t A(t; \tau_i), \tag{29}$$

where $A(t)$ denotes either $A_{\text{safe}}(t)$ or $A_{\text{assist}}(t)$. After sorting $m_{(1)} \le \cdots \le m_{(n)}$, we determine the threshold following common practice in unsupervised anomaly detection, where the decision boundary is estimated from the empirical distribution of normal behaviors. Specifically, we use the 95th percentile of the trajectory-level maximum scores:

$$\gamma_{0.95} = m_{(\lceil 0.95n \rceil)}. \tag{30}$$

During online inference, the detector continuously monitors the execution and raises an alarm at the first step where the detection score exceeds this threshold:

$$t_{\text{alarm}} = \min\{t : A(t) > \gamma_{0.95}\}. \tag{31}$$

If no such step exists, the execution is considered to continue normally. This calibration strategy provides a unified decision rule for both settings while ensuring that the alarm threshold is estimated

exclusively from successful executions. The transition-memory score requires a predecessor, so the fused monitor begins at the second observable execution step. At every scored step, retrieval and prediction use only the task and steps $< t$; the newly observed step is used only in the discrepancy computation. No future step, final outcome, or test label is available to the monitor.

C Additional Experimental Results

C.1 Selection of g_θ and ℓ_{dyn}

We evaluate different choices of the safe-dynamics predictor g_θ and its training objective ℓ_{dyn} by comparing recurrent and attention-based architectures under MSE and cosine objectives. For a controlled comparison, we keep the execution encoder, safe-only data split, optimizer, checkpoint selection rule, and domain-level P95 calibration fixed across all configurations. The Transformer consists of four causal layers with eight attention heads, a hidden dimension of 512, and a feed-forward dimension of 2,048. For each training objective, we use the corresponding prediction discrepancy as the dynamics-based anomaly score.

Table 9: Ablation of the safe-dynamics predictor and training objective in LFS on AFTraj-2K.

LFS configuration	AUROC (%) ↑	AUPRC (%) ↑	Exact-F1 (%) ↑	Step-Acc (%) ↑	ASS ↓
LSTM (MSE)	91.59 ± 0.13	63.43 ± 0.63	45.23 ± 1.10	36.32 ± 0.80	1.202 ± 0.086
LSTM $(1 - \cos)$	$\mathbf{93.15 \pm 0.11}$	$\mathbf{68.36 \pm 0.31}$	$\mathbf{48.38 \pm 1.13}$	41.35 ± 1.27	$\mathbf{1.073 \pm 0.104}$
Transformer (MSE)	91.62 ± 0.35	61.76 ± 0.49	39.76 ± 1.78	32.76 ± 1.66	1.565 ± 0.059
Transformer $(1 - \cos)$	92.19 ± 0.16	66.06 ± 0.55	47.73 ± 0.59	$\mathbf{41.60 \pm 0.91}$	1.140 ± 0.048

As shown in Table 9, the cosine objective consistently outperforms MSE for both predictor architectures, particularly in AUPRC, Exact-F1, and ASS. Between the two architectures, the LSTM with the cosine objective achieves the strongest overall performance, while the Transformer obtains a slightly higher Step-Acc. We therefore adopt the two-layer LSTM with $(1 - \cos)$ as the default safe-dynamics predictor g_θ and training objective ℓ_{dyn} in our experiments.

C.2 Failure-Memory Representation and Aggregation

The failure-assisted score compares the current transition with retrieved safe and failed decisive transitions from three complementary perspectives: semantic state $\mathbf{s}$, transition direction $\mathbf{d}$, and safe-predictor residual $\mathbf{e}$. We ablate these transition representations as well as the number r of candidates retained by the top-r aggregation. All configurations use 50% failure support, ten independently sampled support sets, the adaptive min-gap margin, and P95 calibration based exclusively on safe validation trajectories.

Table 10: Ablation of the failure-assisted transition representation and top-r aggregation on AFTraj-2K. Panel A fixes $r = 5$, while Panel B uses all three transition features.

Panel	Score / aggregation	AUROC (%) ↑	AUPRC (%) ↑	Exact-F1 (%) ↑	Step-Acc (%) ↑	ASS ↓
A: compatibility features	$\mathbf{s}$	93.50±0.46	73.29±1.05	56.51±1.79	49.20±1.75	0.860±0.070
	$\mathbf{d}$	94.47±0.56	77.38±1.30	60.88±2.17	**56.63±2.49**	0.890±0.050
	$\mathbf{e}$	94.10±0.09	75.19±0.28	56.49±0.53	43.80±0.30	**0.320±0.020**
	$\mathbf{s} + \mathbf{d}$	94.78±0.45	77.63±1.32	60.99±2.04	56.20±1.94	0.840±0.090
	$\mathbf{s} + \mathbf{e}$	95.07±0.09	77.41±0.28	59.72±0.67	48.34±0.53	0.540±0.030
	$\mathbf{s} + \mathbf{d} + \mathbf{e}$	**96.03±0.13**	**80.85±0.65**	**61.76±1.05**	52.27±1.22	0.870±0.100
B: top-r mean	$r = 1$	95.91±0.18	79.80±0.68	57.62±1.52	49.63±1.51	0.890±0.050
	$r = 2$	**96.09±0.18**	80.74±0.64	59.26±1.91	50.74±1.62	**0.790±0.090**
	$r = 5$	96.03±0.13	**80.85±0.65**	**61.76±1.05**	**52.27±1.22**	0.870±0.100
	$r = 10$	95.93±0.11	80.12±0.43	59.41±1.08	49.51±0.73	0.830±0.060

As shown in Table 10, the three transition features capture complementary information. Among the individual features, transition direction $\mathbf{d}$ provides the strongest Exact-F1 and Step-Acc, while the predictor residual $\mathbf{e}$ achieves the lowest ASS, indicating more temporally precise localization among detected failures. Combining all three features yields the best AUROC, AUPRC, and Exact-F1, supporting the benefit of jointly modeling semantic state, transition evolution, and deviation from the learned safe dynamics.

Panel B further examines the effect of top-r aggregation. Using a single retrieved transition ($r = 1$) is more sensitive to an individual support example, whereas aggregating multiple highly compatible transitions provides a more stable local comparison. Moderate aggregation performs best overall: $r = 2$ achieves the highest AUROC and lowest ASS, while $r = 5$ achieves the best AUPRC, Exact-F1, and Step-Acc. Increasing r further to 10 slightly degrades most metrics, suggesting that including less relevant retrieved transitions can dilute the most informative local evidence. We therefore use $r = 5$ as the default, as it provides the strongest overall balance between ranking quality and exact failure localization.

C.3 Domain Mixing and Domain Specific Training

By default, LFS pools safe trajectories from all three AFTraj-2K domains when training the safe-dynamics predictor and constructing the safe transition memory. To examine whether cross-domain sharing is beneficial, we independently vary whether the predictor and safe memory are built from mixed-domain or domain-specific safe trajectories, while keeping all other settings unchanged.

Table 11: Comparison of mixed-domain and domain-specific safe modeling on AFTraj-2K.

Evaluation	Predictor	Memory	AUROC (%) ↑	AUPRC (%) ↑	Exact-F1 (%) ↑	ASS ↓	Step-Acc (%) ↑
Overall	Mixed	Mixed	**93.15±0.11**	68.36±0.31	48.38±1.13	1.073±0.104	41.35±1.27
Overall	Mixed	Domain-specific	**93.15±0.11**	**68.37±0.31**	48.21±1.11	1.065±0.099	41.35±1.27
Overall	Domain-specific	Mixed	92.50±0.12	65.87±0.24	49.36±1.17	**1.062±0.100**	**42.21±0.80**
Overall	Domain-specific	Domain-specific	92.50±0.12	65.89±0.24	**49.43±1.16**	**1.062±0.100**	**42.21±0.80**
Coding	Mixed	Mixed	96.66±0.05	79.15±1.17	60.76±3.28	0.566±0.182	64.32±2.26
Coding	Domain-specific	Mixed	**96.71±0.13**	**80.66±0.38**	**65.36±4.48**	**0.451±0.278**	**67.03±2.96**
Math	Mixed	Mixed	**94.91±0.12**	**76.64±0.32**	67.88±1.22	**0.979±0.075**	63.79±2.11
Math	Domain-specific	Mixed	94.61±0.34	76.29±0.40	**68.73±0.81**	1.004±0.054	**65.17±0.77**
Agentic	Mixed	Mixed	**85.14±0.28**	**48.86±1.10**	**14.45±1.67**	**2.242±0.305**	**9.71±1.32**
Agentic	Domain-specific	Mixed	82.31±0.21	42.23±0.27	13.36±0.76	2.333±0.160	9.12±0.66

As shown in Table 11, mixed-domain predictor training provides stronger overall ranking performance, although the effect varies across domains. Coding benefits from domain-specific training, while Math is comparatively insensitive to domain mixing. In contrast, Agentic shows a clear advantage from mixed-domain training, suggesting that shared execution patterns from other domains can provide useful additional supervision when in-domain safe trajectories exhibit more heterogeneous behavior.

The choice between mixed and domain-specific safe memory has substantially less effect. Under the mixed memory, 98.28% of the 3,489 test queries already retrieve a same-domain transition as their top-1 neighbor, including 95.53% for Coding, 99.30% for Math, and 98.84% for Agentic. Thus, similarity-based retrieval naturally favors domain-consistent transitions even without explicit memory partitioning, explaining the nearly identical performance of mixed and domain-specific memories. Based on these results, we use mixed-domain training and a mixed safe memory as the default configuration of LFS.

C.4 Overall Comparison between Failure-Assisted LFS and Fully Supervised Baselines

Table 12 reports the full results on AFTraj-2K across the Math, Coding, and Agentic domains. Following the evaluation protocol in the main body, the safe-dynamics predictor of LFS is trained exclusively on safe trajectories, while different proportions of failed training trajectories are used only to construct the contrastive failure transition memory. In contrast, the supervised baselines use the full available failure supervision. For LFS, results are reported as mean ± standard deviation over repeated failure-support sampling.

Table 12: Overall comparison between failure-assisted LFS and fully supervised methods on AFTraj-2K. The **top-1** and **top-2** values are highlighted.

Method	Usage of Failed Support	Math		Coding		Agentic		Overall	
		Exact-F1(%) ↑	ASS ↓	Exact-F1(%) ↑	ASS ↓	Exact-F1(%) ↑	ASS ↓	Exact-F1(%) ↑	ASS ↓
Open-source LLM auditors									
Llama3.2-3B	100%	8.14	4.86	21.05	2.94	16.13	2.30	14.41	3.38
Gemma3-4B	100%	1.15	5.78	12.90	4.59	8.29	3.04	6.92	4.37
Qwen2.5-7B-Instruct	100%	10.39	3.80	38.20	2.26	14.00	2.96	21.05	2.75
Qwen3-8B	100%	21.95	4.65	27.85	2.57	33.64	1.57	28.36	2.65
Qwen3-32B	100%	18.63	4.07	20.00	2.83	40.00	1.59	26.91	2.82
Proprietary LLM auditors									
GPT-4.1	100%	24.39	3.95	10.81	3.50	40.68	1.29	27.43	2.67
Gemini-3-Flash	100%	40.52	2.48	19.42	2.73	26.09	1.70	29.74	2.19
Claude-Haiku-4.5	100%	19.75	4.12	23.91	2.80	31.95	1.69	25.53	2.85
DeepSeek-V4-Flash	100%	42.77	2.78	38.10	1.27	32.53	1.42	37.65	1.94
DeepSeek-V4-Pro	100%	50.34	2.60	49.32	0.96	**41.77**	1.31	46.56	1.77
Qwen2.5-7B-Instruct-based methods									
Perplexity-7B	100%	2.31	4.09	26.56	2.19	16.57	2.50	14.11	3.02
ToT-7B	100%	20.38	4.40	7.02	4.06	24.84	2.13	18.52	3.39
Reflexion-7B	100%	16.57	4.39	9.52	4.50	39.13	1.44	23.38	3.17
AgentDebug-7B	100%	0.00	–	28.57	4.05	2.82	**1.00**	9.63	3.76
AgentForesight-7B	100%	77.36	0.96	78.87	**0.18**	**48.70**	**0.54**	**66.44**	**0.59**
Safe-only LFS (Qwen2.5-3B-Instruct)	0%	67.88±1.22	0.979±0.075	60.76±3.28	**0.566**±0.182	14.45±1.67	2.242±0.305	48.38±1.13	1.073±0.104
Failed trajectories used as contrastive transition memory									
Failure-Assisted LFS (Qwen2.5-3B-Instruct)	5%	75.11±6.55	0.721±0.424	68.49±0.81	1.200±0.015	10.63±4.19	2.618±0.650	53.15±2.93	1.143±0.222
	10%	83.78±4.07	0.368±0.252	69.29±0.84	1.183±0.017	11.44±2.56	2.421±0.205	57.10±1.25	0.964±0.118
	15%	85.37±1.58	0.323±0.133	70.60±2.29	1.149±0.055	12.08±1.55	2.305±0.172	58.26±0.98	0.921±0.061
	20%	85.87±0.31	**0.310**±0.144	74.26±3.27	1.058±0.076	12.82±1.77	2.247±0.190	59.73±1.13	0.875±0.066
	25%	85.72±0.34	0.325±0.122	74.59±3.52	1.050±0.084	12.85±1.96	2.202±0.157	59.81±1.08	0.874±0.080
	30%	85.87±0.31	0.318±0.145	76.26±4.05	1.012±0.090	13.50±1.70	2.141±0.118	60.43±0.89	0.860±0.077
	35%	85.87±0.31	**0.290**±0.083	78.08±3.22	0.967±0.071	14.33±1.27	2.136±0.072	61.11±0.73	0.838±0.046
	40%	**86.02**±0.51	0.335±0.154	**80.32**±2.44	0.924±0.043	16.43±3.45	1.999±0.194	**62.15**±0.76	**0.835**±0.075
	45%	**86.02**±0.51	0.376±0.185	**79.89**±2.17	0.924±0.043	14.26±1.91	2.100±0.141	61.39±0.77	0.870±0.084
	50%	85.72±0.68	0.395±0.212	79.83±2.08	0.929±0.042	15.97±2.82	2.029±0.197	61.76±1.05	0.869±0.096