DREAM2DISCERN: LEARNING FROM HARD LATENT VARIANTS FOR RUNTIME FAILURE PREDICTION IN MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reliable deployment of LLM-based multi-agent systems calls for runtime auditors that predict terminal outcomes from observed execution prefixes. These auditors typically learn from a limited corpus of trajectories with verified outcomes. Expanding this corpus requires costly execution and verification, yet broader failure coverage alone provides no explicit criterion for identifying prefixes that the current auditor finds difficult to classify. To this end, we propose ***Dream2Discern***, a runtime auditor trained through implicit augmentation guided by its own prediction loss. During *Dream*, we select perturbations of the auditor's representation operators to increase prediction loss on recorded prefixes, yielding *hard latent variants*. During *Discern*, the auditor learns to predict execution outcomes from these variants using the original labels. The variants are regenerated as the auditor learns, turning a limited corpus into a renewable source of hard examples and requiring no additional trajectory collection or outcome verification. Extensive evaluations on AFTRAJ-2K and StepShield show that *Dream2Discern* outperforms competitive failure auditors and achieves over 90% trajectory-level weighted accuracy on both benchmarks while balancing prediction accuracy and timeliness.

1 INTRODUCTION

LLM-based multi-agent systems coordinate specialized agents and tools to solve complex tasks, from software development to open-ended web interaction (Wu et al., 2024; Hong et al., 2023; Fourney et al., 2024). However, the complexity of these tasks and their execution environments makes agents prone to reasoning and tool-use errors that can potentially compromise task completion (Cemri et al., 2026; Zhang et al., 2025b; Zhu et al., 2025). In open-world environments, these errors can lead agents to make unnecessary tool calls or modify external systems in unintended ways (Ruan et al., 2024; Oh et al., 2026). Accordingly, reliable deployment calls for *runtime failure prediction*, where an auditor assesses the risk of eventual task failure from the execution history observed so far to inform timely intervention (Zhang et al., 2026; Huang et al., 2026).

Despite its importance, existing learning-based runtime auditors, such as TRACES (Li et al., 2026) and AgentForesight (Zhang et al., 2026), typically rely on a limited corpus of carefully curated trajectories with verified outcomes, as illustrated in Figure 1(a). A straightforward way to improve these auditors is to augment the training corpus to cover a broader range of failure scenarios, either by collecting new trajectories or by editing existing ones (Figure 1(b)). However, collecting new trajectories requires agent rollouts and outcome verification, making corpus expansion costly to scale (Xie et al., 2026; Xu et al., 2025). Editing recorded trajectories reuses the observed history, but changing an action or tool response can alter the continuation and invalidate the original outcome label, requiring resumed execution and re-verification (Kong et al., 2026; Bonagiri et al., 2026). More fundamentally, broader failure coverage alone provides no explicit criterion for identifying trajectories whose prefixes the current auditor finds difficult to assess. These considerations naturally motivate the following research question: ***Can we identify and exploit the current auditor's hard cases without acquiring additional labeled trajectories?***

To address this question, we propose an implicit augmentation approach that constructs hard latent variants within the auditor (Figure 1(c)). **Our key insight is to use the auditor's own prediction**

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

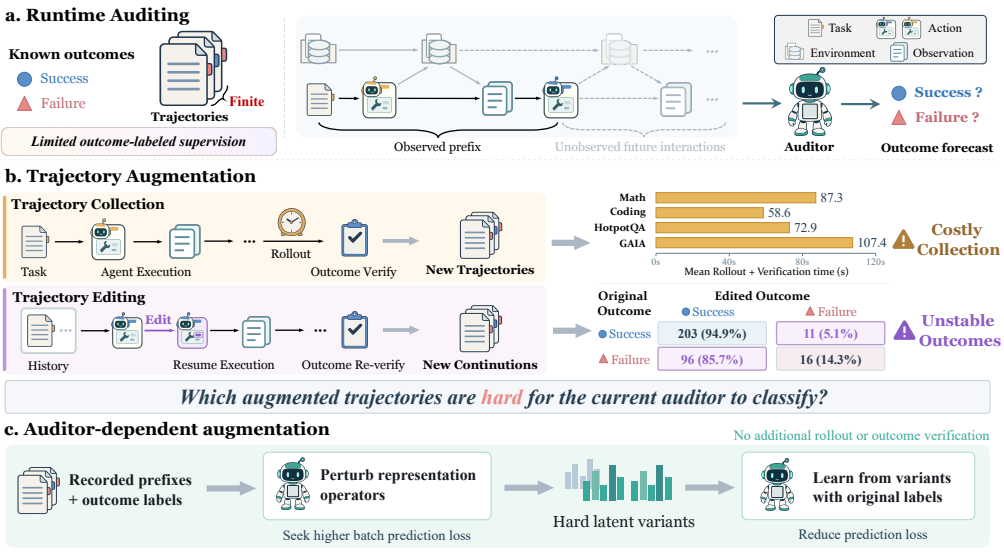


Figure 1: **Motivation of *Dream2Discern*. (a)** A runtime auditor predicts terminal outcomes from observed prefixes but learns from a limited labeled corpus. **(b)** Trajectory augmentation seeks broader failure coverage, but collection is costly and editing can invalidate outcome labels (right). Broader coverage alone does not identify prefixes that the current auditor finds hard to classify. **(c)** We construct hard latent variants through operator perturbations chosen to increase batch prediction loss, then train on these variants while keeping recorded prefixes and labels unchanged.

loss as a criterion for deciding which variants to construct. We perturb the auditor's representation computation to increase prediction loss on recorded prefixes, yielding *hard latent variants* for training with the original outcome labels. To realize this idea, we develop ***Dream2Discern***, a runtime auditor that alternates between two training steps: constructing hard latent variants (*Dream*) and learning from them to discern execution outcomes (*Discern*). Specifically, during ***Dream***, we search for bounded, low-rank perturbations to the auditor's query and value projection weights that increase batch prediction loss on recorded prefixes. We keep the prediction head fixed during this search, thus restricting the induced difficulty to changes in prefix representations. In ***Discern***, we train the auditor to distinguish success from failure under the selected perturbation. At runtime, the trained auditor operates without perturbations, assigning a failure score to each observed prefix and applying a calibrated threshold to predict the terminal outcome. Together, the *Dream* and *Discern* steps alternate throughout training, allowing the constructed variants to evolve with the auditor.

We conduct extensive runtime evaluations of *Dream2Discern* on AFTRAJ-2K (Zhang et al., 2026) and StepShield (Felicia et al., 2026). Our method achieves over $\mathbf{90}\%$ trajectory-level weighted accuracy (wACC) on both benchmarks and outperforms the strongest auditor baseline by up to $\mathbf{11.11}\%$ wACC, while maintaining a practical accuracy–timeliness trade-off. These gains extend to Llama backbones, and loss-guided variants improve outcome discrimination over training without perturbations or with random perturbations. Our main contributions are summarized as follows:

- We show that augmenting the training corpus with edited prefixes does not improve held-out prediction under a fixed training budget. This motivates constructing hard latent variants tailored to the current auditor (Section 3.1).
- We develop ***Dream2Discern***, a parameter-efficient runtime auditor with alternating *Dream* and *Discern* updates. *Dream* constructs hard latent variants through loss-guided perturbations of the auditor's representation computation, and *Discern* learns to predict execution outcomes from these variants using the recorded labels (Section 3.2 and Section 3.3).
- We demonstrate improved prediction performance on AFTRAJ-2K and StepShield across different model families. Ablations and further analyses support the effectiveness of loss-guided variant construction and its contribution to runtime auditor learning (Section 4).

2 PROBLEM FORMULATION

Multi-Agent Trajectory. Consider a system of LLM agents that collaborate on a task through communication and tool use. For execution i, let c_i denote the initial task context available to the auditor. We write the complete trajectory as $\tau_i = (c_i, u_{i,1}, \ldots, u_{i,T_i})$, where T_i is the number of interaction steps. Each record $u_{i,t}$ identifies its source (agent, user, tool, environment) and records the associated message, action, or observation. Each completed trajectory has a terminal outcome $Y_i \in \{0, 1\}$, where $Y_i = 1$ denotes failure and $Y_i = 0$ denotes success.

Runtime Failure Prediction. Following prior work on runtime monitoring (Zhang et al., 2026; Li et al., 2026; Park et al., 2026), we consider an auditor that observes only the prefix $\tau_{i,\le t} = (c_i, u_{i,1}, \ldots, u_{i,t})$ at each nonterminal step $1 \le t < T_i$. The auditor predicts the terminal outcome Y_i from this prefix, with its prediction denoted by $\widehat{Y}_{i,t} \in \{0, 1\}$. As illustrated in Figure 1(a), the auditor updates its prediction after each newly observed interaction. Future interactions, the realized terminal length T_i, and the outcome Y_i are unavailable to the auditor at prediction time.

Learning Setup. Let f_Θ denote an auditor with trainable parameters Θ and a sigmoid output. Given an input $X_{i,t}$ constructed from the observed prefix $\tau_{i,\le t}$, it produces a scalar failure score

$$s_{i,t} = f_\Theta(X_{i,t}) \in [0, 1]. \tag{1}$$

Thresholding this score yields $\widehat{Y}_{i,t}$, with the decision rule specified in Section 3.4. For the learning objective, let $q = (i, t)$ index a recorded prefix with input $X_q = X_{i,t}$ and the corresponding trajectory's terminal outcome $Y_q = Y_i$ as its prediction target. For an indexed batch $\mathcal{B} = (q_b)_{b=1}^n$ of selected prefixes, we define the mean BCE loss as $\mathcal{L}_\mathcal{B}(\Theta) = \frac{1}{n}\sum_{b=1}^n \ell_{\mathrm{BCE}}(f_\Theta(X_{q_b}), Y_{q_b})$.

3 METHOD

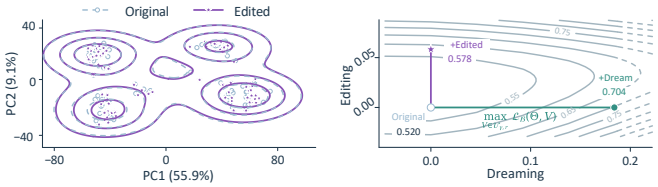
We present ***Dream2Discern***, a runtime auditor trained through alternating *Dream* and *Discern* steps. We first analyze the limitations of trajectory augmentation and motivate prediction loss as a criterion for constructing hard latent variants. We then describe *Dream*, which constructs hard latent variants by perturbing the auditor's adapted representation path. The *Discern* step trains the auditor under the selected perturbation. Finally, we describe how prefix failure scores yield runtime decisions.

3.1 MOTIVATION

Reliable deployment of LLM-based multi-agent systems calls for runtime failure prediction to support timely intervention. Existing learning-based auditors typically learn from a limited corpus of carefully curated trajectories with verified outcomes. To improve prediction, a straightforward approach is to augment this corpus to cover a broader range of failure scenarios (Xu et al., 2025; Kong et al., 2026). However, the practical value of this approach depends on both the cost of acquiring additional labeled trajectories and whether they improve held-out prediction.

To this end, we first assess the cost of acquiring additional labeled trajectories. We measure the combined rollout and outcome-verification time for $\sim$2K trajectories across task domains in AFTRAJ-2K (Zhang et al., 2026) (see the right panel of Figure 1(b)). Results show that averaged acquisition times reach tens of seconds per trajectory, making repeated corpus expansion time-consuming and unsustainable. Editing offers an alternative by reusing recorded history, but the resulting continuation may no longer share the original outcome label. To examine this issue, we modify an action or tool response and resume execution, obtaining 326 verified continuations from 97 source trajectories in AFTRAJ-2K (see Appendix E.1.1). Among verified continuations from failed source trajectories, 85.7% succeed, showing that source labels cannot simply be reused after editing and re-execution.

Beyond acquisition cost, we examine whether the verified edits improve auditor learning. We first compare paired original and edited prefix representations extracted by the same frozen LLM encoder. Despite their textual differences, the two groups substantially overlap in a shared PCA projection (Figure 2(a)). To evaluate their training utility, we train prediction heads on original prefixes alone (*Original*) or together with the verified edits (*+Edited*), keeping the encoder and optimization budget fixed. We evaluate both heads on the same held-out prefixes using class-balanced BCE, where lower loss indicates better prediction. The higher loss of *+Edited* relative to *Original* (Figure 2(b)) shows that these edits do not improve held-out prediction under this budget.



(a) Representation overlap **(b) Held-out prediction loss**

Figure 2: **(a)** Original and edited prefixes overlap in a shared PCA projection. **(b)** Training with edits does not reduce held-out class-balanced BCE. A temporary *Dream* perturbation selected on training prefixes raises this loss with the *Original* head fixed.

These observations motivate us to construct hard latent variants that target the auditor's prediction difficulties. We propose to seek bounded weight perturbations that alter prefix representations toward higher batch prediction loss, while preserving the recorded prefixes and outcome labels. To examine this intuition, we select a perturbation using only training prefixes and temporarily apply it to the encoder, keeping the *Original* prediction head fixed. On the same held-out prefixes, the perturbed auditor (+*Dream*) has higher BCE than *Original* (Figure 2(b)). This result shows that the loss-guided perturbation also makes outcome prediction harder beyond the batch used to select it, without acquiring additional labeled trajectories. Experimental setup details are in Appendix E.1.2.

3.2 DREAM: CONSTRUCTING HARD LATENT VARIANTS

We instantiate f_Θ with a pretrained LLM (e.g., Qwen3-8B) whose weights ϕ_0 remain frozen. We apply LoRA (Hu et al., 2021) to the query and value projections in Transformer blocks following a fixed intermediate block, and denote the adapter parameters by a. Let $h_{\phi_0,a}(X)$ denote the final-layer hidden state at the *last token* of prefix input X. A classification head g_ψ maps this state to a scalar logit, yielding $f_\Theta(X) = \sigma(g_\psi(h_{\phi_0,a}(X)))$, where σ is the sigmoid function and $\Theta = (a, \psi)$.

Each training batch $\mathcal{B} = (q_b)_{b=1}^{2m}$ contains m prefixes from each outcome class, so $n = 2m$ in the batch BCE loss (Figure 3, ①). In the *Dream* step, we hold Θ fixed and seek a shared perturbation that increases prediction loss on this batch. The perturbation acts only on the effective weights of the adapted query and value projections, so the search changes prefix representations under the fixed prediction head. Let $\mathcal{S}$ index these projections and $V = (V_j)_{j\in\mathcal{S}}$ denote the perturbation shared across the batch. The effective projection weights and their temporary perturbations are

$$W_j(a) = W_j^0 + \kappa_L B_j A_j, \qquad W_j(a) \;\mapsto\; W_j(a) + V_j, \qquad j \in \mathcal{S}, \tag{2}$$

where W_j^0 is frozen, A_j and B_j are the trainable LoRA factors collected in a, and κ_L is the adapter scaling. Since $B_j A_j = (cB_j)(A_j/c)$ for any $c \neq 0$, an operator-level perturbation budget is invariant to this rescaling of the LoRA factors. We write $\mathcal{L}_\mathcal{B}(\Theta, V)$ for the batch loss in Section 2 computed with these perturbed operators, with $\mathcal{L}_\mathcal{B}(\Theta, 0) = \mathcal{L}_\mathcal{B}(\Theta)$. Throughout the *Dream* step, the recorded prefix inputs and their outcome labels remain unchanged.

Bounded, low-rank perturbations. To identify prediction difficulties under small, structured changes to the auditor's operators, we bound the relative perturbation magnitudes and the rank at each projection. Using frozen projection norms $\omega_j := \|W_j^0\|_F > 0$ as reference scales, we define:

$$\|V\|_\omega^2 := \sum_{j\in\mathcal{S}} \frac{\|V_j\|_F^2}{\omega_j^2}, \quad \mathcal{U}_{\gamma,r} := \big\{V : \; \|V\|_\omega \le \gamma, \quad \mathrm{rank}(V_j) \le r \;\; \forall j \in \mathcal{S}\big\}, \tag{3}$$

where $\gamma > 0$ is a global relative radius and $r \ge 1$ is an integer rank cap. The scales normalize perturbation magnitudes across projections, while the single budget bounds their combined relative size. We keep these scales fixed so that adapter updates do not change the reference used to measure perturbation size. The rank cap limits each projection's direct output displacement up to r directions.

From operator perturbations to hard latent variants. Within the perturbation set above, we define the ideal *Dream* perturbation by maximizing batch prediction loss:

$$V_\mathcal{B}^\star(\Theta) \in \arg\max_{V\in\mathcal{U}_{\gamma,r}} \mathcal{L}_\mathcal{B}(\Theta, V). \tag{4}$$

For an individual prefix, BCE increases when the predicted failure score rises for a successful trajectory or falls for a failed one. The objective gives both outcome classes equal weight and evaluates a single perturbation on the batch. Under the fixed prediction head, *Dream* seeks latent variants of the

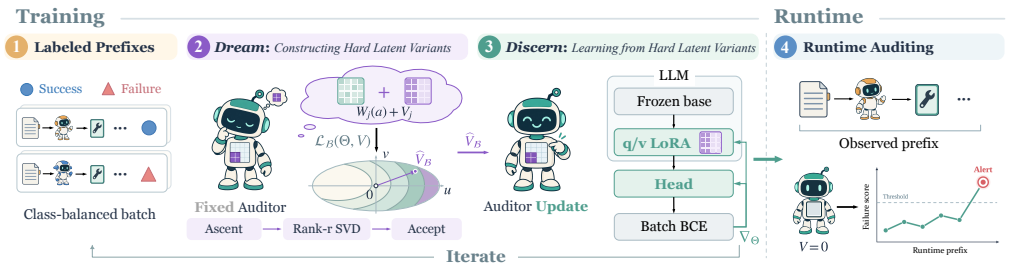


Figure 3: **Overview of *Dream2Discern*. ① *Labeled prefixes*.** We form a class-balanced batch from recorded success and failure prefixes, keeping their inputs and outcome labels unchanged throughout the training steps. **② *Dream*.** With the auditor fixed, we search for shared, bounded, low-rank perturbations to its query and value projection weights that increase batch prediction loss. **③ *Discern*.** Holding the selected perturbation fixed, we update the LoRA adapters and prediction head to minimize the loss on the resulting hard latent variants. **④ *Runtime auditing*.** We discard the perturbation and apply a calibrated threshold to each prefix score to predict failure during execution.

recorded prefixes with higher batch prediction loss. Although the operator perturbation is shared, its effect depends on the incoming token states of each prefix (Wang et al., 2023). For a fixed V, let $H^V_{q,j}$ collect the token states entering projection j for prefix q in the perturbed forward pass. With the compact SVD $V_j = P_j \Sigma_j Q_j^\top$, the direct additive contribution of V_j at these incoming states is

$$H^V_{q,j} V_j^\top = (H^V_{q,j} Q_j)\, \Sigma_j\, P_j^\top. \tag{5}$$

The columns of P_j define at most r output directions shared across prefixes, while $H^V_{q,j} Q_j \Sigma_j$ determines their prefix-dependent coefficients. Query perturbations can alter attention weights over prefix tokens, while value perturbations can alter the vectors aggregated by attention. These local changes propagate through the remaining Transformer computation to the final prefix representations. The final prefix representations induced by an accepted *Dream* perturbation form *hard latent variants*, with hardness measured by the increase in batch prediction loss. *Discern* then learns to predict the recorded outcomes from these variants. We further observe in Section 4.4 that held-out prefixes with larger absolute score changes under training-selected *Dream* perturbations are more often misclassified by the unperturbed auditor.

Solving for the perturbation. To construct hard latent variants, we seek operator changes within $\mathcal{U}_{\gamma,r}$ that maximize the first-order increase in batch prediction loss. Hence, we linearize the objective in Equation 4 at the unperturbed auditor with $V = 0$. Let $G_j = \nabla_{V_j} \mathcal{L}_\mathcal{B}(\Theta, V)\big|_{V=0}$ denote the loss gradient for projection j. The first-order loss increase is $\sum_{j\in\mathcal{S}} \langle G_j, V_j \rangle$, where $\langle A, B \rangle = \mathrm{tr}(A^\top B)$ is the Frobenius inner product. In the scaled coordinates of Equation 3, the gradient is $C_j = \omega_j G_j$. Let $C_{j,r}$ be a best approximation of C_j of rank at most r in Frobenius norm, obtained by SVD truncation. Define $g_r = (\sum_{j\in\mathcal{S}} \|C_{j,r}\|_F^2)^{1/2}$ as the norm of the retained scaled gradients.

Proposition 1 (Optimal linearized perturbation)**.** *For fixed Θ and $\mathcal{B}$, the maximum of $\sum_{j\in\mathcal{S}} \langle G_j, V_j \rangle$ over $\mathcal{U}_{\gamma,r}$ is γg_r. If $g_r > 0$, one maximizer is*

$$V_j^{\mathrm{lin}} = \gamma\, \omega_j\, \frac{C_{j,r}}{g_r}, \qquad j \in \mathcal{S}. \tag{6}$$

If $g_r = 0$, the maximum is attained at $V = 0$.

The proof is given in Appendix C.1. The solution retains the leading singular components of each scaled gradient and allocates relative perturbation magnitudes in proportion to their retained Frobenius norms. In practice, we normalize the full scaled gradient before SVD truncation (Figure 3, ②). For $g_r > 0$, the resulting candidate $\overline{V}_\mathcal{B}$ follows one such maximizing direction, with relative radius at most γ, while it is zero for $g_r = 0$. To check whether the candidate increases the nonlinear batch loss, we start at $\overline{V}_\mathcal{B}$ and successively halve it, accepting the first tested point whose measured loss strictly exceeds the loss at $V = 0$. We test at most six points, as specified in Algorithm 1 of Appendix B.1. If no tested step is accepted, we return $V = 0$. Let $\lambda_\mathcal{B} \in \{0, 1, 2^{-1}, \ldots, 2^{-5}\}$ denote the returned step fraction, with 0 indicating rejection. The returned perturbation and realized radius:

$$\widehat{V}_\mathcal{B} = \lambda_\mathcal{B} \overline{V}_\mathcal{B}, \qquad \rho_\mathcal{B} := \|\widehat{V}_\mathcal{B}\|_\omega. \tag{7}$$

Each batch starts a fresh search from $V = 0$ without carrying perturbations between batches.

3.3 DISCERN: LEARNING FROM HARD LATENT VARIANTS

The *Discern* step trains the auditor on the prefix representations induced by the perturbation $\widehat{V}_{\mathcal{B}}$ returned by the *Dream* step as depicted in Figure 3, ③. Each prefix retains its recorded trajectory outcome Y_q as the prediction target, since the *Dream* step changes the auditor's representation computation while keeping the input X_q unchanged. To learn from these representation changes, we hold $\widehat{V}_{\mathcal{B}}$ fixed and take an optimization step using only the perturbed batch loss:

$$\min_{\Theta} \; \mathcal{L}_{\mathcal{B}}\Big(\Theta, \mathrm{sg}[\widehat{V}_{\mathcal{B}}]\Big), \tag{8}$$

where sg stops gradients through the *Dream* search. Gradients of this loss pass through the perturbed forward pass to update the adapters a and classification head ψ, while the pretrained weights ϕ_0 remain frozen. If the *Dream* step returns zero, Equation 8 uses the unperturbed batch loss $\mathcal{L}_{\mathcal{B}}(\Theta)$.

Interpreting the *Discern* loss. To explain what hard latent variants add to training on the recorded prefixes, we relate the loss used by *Discern* to the unperturbed prediction loss. The following result connects their difference to the current auditor's sensitivity to the permitted low-rank operator changes and the perturbation radius returned by *Dream*.

Proposition 2 (Loss at the returned perturbation)**.** *Fix Θ and $\mathcal{B}$, and consider the Dream step in Algorithm 1 in exact arithmetic. Assume that, for some $\beta \geq 0$, the first-order Taylor remainder of $\mathcal{L}_{\mathcal{B}}(\Theta, V)$ at $V = 0$ has magnitude at most $\frac{\beta}{2}\|V\|_{\omega}^2$ for every V on the segment from 0 to $\overline{V}_{\mathcal{B}}$. Then*

$$\Big|\mathcal{L}_{\mathcal{B}}(\Theta, \widehat{V}_{\mathcal{B}}) - \mathcal{L}_{\mathcal{B}}(\Theta, 0) - \rho_{\mathcal{B}} g_r\Big| \leq \frac{\beta}{2}\rho_{\mathcal{B}}^2. \tag{9}$$

To first order, the loss evaluated by *Discern* is the unperturbed prediction loss plus $\rho_{\mathcal{B}} g_r$. By Proposition 1, g_r is the largest first-order loss increase per unit relative radius under the rank constraints. The additional loss therefore reflects the current auditor's sensitivity to these operator changes, with $\rho_{\mathcal{B}}$ accounting for the step actually returned by *Dream*. The remainder bounds the nonlinear deviation from this local interpretation. The proof is given in Appendix C.2, and implementation details are in Appendix B.1. Repeating *Dream* and *Discern* trains f_Θ to predict outcomes from variants selected using its current loss. At runtime, the resulting auditor uses the learned LoRA adapters and prediction head with $V = 0$.

3.4 RUNTIME AUDITING

At each nonterminal interaction step, the trained auditor assigns a failure score to the observed prefix and predicts the terminal outcome (Figure 3, ④):

$$s_{i,t} = f_\Theta(X_{i,t}), \qquad \widehat{Y}_{i,t} = \mathbf{1}\{s_{i,t} \geq \zeta_t\}, \qquad 1 \leq t < T_i. \tag{10}$$

Each prediction requires one auditor forward pass and a scalar threshold comparison. To account for variation in score distributions across interaction steps, we fit step-dependent thresholds from successful trajectories (Gu et al., 2026). We first express each calibration score as its empirical percentile at the corresponding step. We then summarize each trajectory by its maximum percentile over nonterminal steps, an aggregation closely related to the max-rank procedure (Timans et al., 2023). A high quantile of these maxima defines a shared rank cutoff which is mapped back to the score scale. The thresholds are fitted offline using successful trajectories from the training partition and remain fixed at runtime. Appendix B.2 gives the complete construction and calibration protocol.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUPS

Benchmarks. We evaluate runtime auditing on AFTRAJ-2K (Zhang et al., 2026) and StepShield (Felicia et al., 2026). AFTRAJ-2K contains multi-agent executions spanning mathematical reasoning, coding, and tool use, while StepShield provides annotated agent trajectories for safety monitoring. We train *Dream2Discern* separately for each benchmark and evaluate it on the corresponding test partition. The test split of AFTRAJ-2K comes from the same corpus as the training split and provides in-domain evaluation, whereas the test trajectories of StepShield come from incident templates never used in training and provide out-of-domain evaluation at the template level.

Table 1: Main comparison on **AFTRAJ-2K** (Zhang et al., 2026) and **StepShield** (Felicia et al., 2026). The best and second-best results in each column are shown in **bold** and underline.

		AFTRAJ-2K				StepShield			
	Method	**AUROC↑**	**FPR95↓**	**wACC↑**	**TWA↑**	**AUROC↑**	**FPR95↓**	**wACC↑**	**TWA↑**
Open.	Qwen3-4B	69.63	86.39	55.12	51.65	81.07	82.41	50.46	49.12
	Qwen3-8B	67.18	80.47	56.93	52.83	84.95	88.89	62.96	53.08
	Qwen3-32B	69.79	82.25	56.33	52.70	90.70	70.37	58.80	52.31
	Llama3.2-3B-it	59.60	90.53	54.22	51.21	78.88	86.11	50.93	50.28
	Llama3.1-8B-it	58.02	94.08	53.01	51.49	83.49	76.85	50.00	49.56
	LlamaGuard3-8B	57.45	94.08	54.22	50.07	83.02	86.11	75.00	56.05
Closed.	GPT-5.4	50.53	99.41	45.48	37.53	88.30	58.33	65.74	41.90
	Claude-Sonnet-4.6	70.90	100.00	63.86	54.83	**95.58**	**38.89**	87.04	<u>56.79</u>
	Grok-4.3	71.57	95.27	65.96	52.34	91.48	58.33	82.87	54.31
	Kimi-K2.7-Code	73.57	100.00	68.67	54.43	90.50	54.07	85.19	54.38
Token Unc.	Max Entropy-8B	53.58	95.86	49.10	47.92	53.70	90.74	49.54	48.87
	Avg. Entropy-8B	66.20	90.53	49.70	47.26	57.54	91.67	50.46	49.46
	Max NLL-8B	45.20	97.63	51.20	48.27	51.58	93.52	50.00	49.02
	Avg. NLL-8B	48.09	99.41	54.52	50.48	57.58	87.96	50.46	50.20
Failure Auditors	AgentDebug-8B	56.78	87.57	51.81	48.73	75.77	85.19	58.33	51.38
	AgentForesight-7B	83.75	89.94	77.11	59.33	83.37	77.78	79.17	56.13
	SAFE-Qwen3-4B	$94.76^{\pm 0.39}$	$41.42^{\pm 3.13}$	$84.74^{\pm 1.94}$	$68.83^{\pm 0.81}$	$84.47^{\pm 0.72}$	$63.27^{\pm 6.17}$	$66.20^{\pm 5.09}$	$52.40^{\pm 1.71}$
	SAFE-Qwen3-8B	$92.69^{\pm 0.81}$	$55.62^{\pm 11.34}$	$85.74^{\pm 1.22}$	$66.57^{\pm 0.39}$	$79.45^{\pm 3.01}$	$63.89^{\pm 10.43}$	$61.73^{\pm 6.96}$	$50.67^{\pm 1.55}$
	H&S-Qwen3-4B	$90.84^{\pm 1.26}$	$72.98^{\pm 11.96}$	$88.76^{\pm 1.14}$	$70.27^{\pm 0.43}$	$85.02^{\pm 0.96}$	$86.42^{\pm 5.66}$	$70.99^{\pm 2.83}$	$54.74^{\pm 1.33}$
	H&S-Qwen3-8B	$89.54^{\pm 0.57}$	$89.15^{\pm 6.73}$	$87.45^{\pm 1.42}$	$69.01^{\pm 1.42}$	$81.58^{\pm 2.56}$	$81.17^{\pm 7.54}$	$69.75^{\pm 0.96}$	$53.82^{\pm 0.69}$
	TRACES-Qwen3-4B	$91.98^{\pm 1.18}$	$60.75^{\pm 16.25}$	$88.35^{\pm 1.14}$	$69.90^{\pm 1.38}$	$85.37^{\pm 2.72}$	$59.88^{\pm 7.98}$	$64.66^{\pm 2.28}$	$52.45^{\pm 2.52}$
	TRACES-Qwen3-8B	$92.63^{\pm 0.64}$	$65.88^{\pm 12.32}$	$87.05^{\pm 1.04}$	$69.91^{\pm 0.48}$	$79.25^{\pm 1.86}$	$79.63^{\pm 12.04}$	$70.83^{\pm 2.12}$	$48.52^{\pm 2.66}$
	Dream2Discern-Qwen3-4B	<u>95.45</u>$^{\pm 0.16}$	<u>38.86</u>$^{\pm 7.81}$	<u>90.96</u>$^{\pm 0.60}$	<u>72.61</u>$^{\pm 0.38}$	$93.62^{\pm 0.68}$	<u>42.28</u>$^{\pm 2.33}$	$\mathbf{90.28}^{\pm 0.46}$	$\mathbf{57.67}^{\pm 0.35}$
	Dream2Discern-Qwen3-8B	$\mathbf{95.82}^{\pm 0.43}$	$\mathbf{31.95}^{\pm 3.07}$	$\mathbf{91.47}^{\pm 0.17}$	$\mathbf{73.45}^{\pm 0.71}$	<u>93.95</u>$^{\pm 0.63}$	$53.09^{\pm 8.40}$	<u>87.50</u>$^{\pm 0.93}$	$55.69^{\pm 1.10}$

Baselines. We compare *Dream2Discern* with four categories of baselines. **(1) Open-weight LLM judges** include Qwen3-4B/8B/32B (Yang et al., 2025), Llama-3.2-3B-Instruct, Llama-3.1-8B-Instruct (Grattafiori et al., 2024), and LlamaGuard3-8B (Inan et al., 2023). **(2) Closed-source LLM judges** include GPT-5.4, Claude-Sonnet-4.6, Grok-4.3, and Kimi-K2.7-Code. **(3) Token uncertainty baselines** use maximum or average token entropy (Malinin & Gales, 2020) and negative log-likelihood (Ren et al., 2022) under a Qwen3-8B proxy model to score the recorded prefixes. **(4) Failure auditors** include AgentDebug (Zhu et al., 2025), AgentForesight (Zhang et al., 2026), SAFE (Gu et al., 2026), Hide-and-Seek (Park et al., 2026), and TRACES (Li et al., 2026).

Evaluation metrics. We evaluate every nonterminal prefix and aggregate predictions at the trajectory level. Using each trajectory's maximum prefix score, we report **AUROC** for outcome discrimination and **FPR95** for the false-positive rate at 95% failure-detection recall. **Weighted accuracy (wACC)** measures trajectory-level classification accuracy under the observed class proportions. **Time-weighted accuracy (TWA)** rewards earlier detection of failed trajectories while accounting for missed failures and false alarms (Römer et al., 2026). For wACC and TWA, we use fitted thresholds at nominal $\alpha = 0.05$, except for proprietary judges whose returned labels define the decisions.

Implementation details. We instantiate *Dream2Discern* on Qwen3-4B, Qwen3-8B, Llama-3.2-3B-Instruct, and Llama-3.1-8B-Instruct. We optimize the LoRA adapters and prediction head with AdamW. A 100-step supervised warm start is followed by a 4-epoch training budget for *Dream* and *Discern*. On Qwen3, we report means and standard deviations over three training seeds for *Dream2Discern*, SAFE, Hide-and-Seek, and TRACES. More details are provided in Appendix D.3.

We provide detailed experimental setups in Appendix D.

4.2 MAIN EXPERIMENTS

Comparison with baseline methods. Table 1 shows that *Dream2Discern* leads all four metrics on AFTRAJ-2K and wACC and TWA on StepShield. LLM judges are more competitive on StepShield, while token uncertainty-based methods provide limited outcome discrimination on both benchmarks. For both backbones, *Dream2Discern* outperforms competitive failure auditor baselines SAFE, Hide-and-Seek, and TRACES on all four metrics averaged over three seeds. Its wACC advantage is substantially larger on out-of-domain StepShield than on in-domain AFTRAJ-2K, supporting our strategy of learning from hard latent variants without collecting additional trajectories. Our 4B auditor also exceeds Claude-Sonnet-4.6 in StepShield wACC and TWA, demonstrating ef-

Table 2: Comparison on Llama backbones.

Benchmark	Base Model	Method	AUROC↑	wACC↑	TWA↑
AFTRAJ-2K	Llama3.2-3B-it	SAFE	$94.62^{\pm 0.09}$	$85.54^{\pm 4.64}$	$69.12^{\pm 2.94}$
		H&S	$91.46^{\pm 0.67}$	$83.23^{\pm 4.52}$	$61.98^{\pm 5.25}$
		TRACES	$91.88^{\pm 0.57}$	$85.54^{\pm 4.44}$	$66.73^{\pm 4.30}$
		Dream2Discern	$\mathbf{95.62}^{\pm 0.61}$	$\mathbf{91.27}^{\pm 0.30}$	$\mathbf{73.38}^{\pm 0.18}$
	Llama3.1-8B-it	SAFE	$94.64^{\pm 0.37}$	$86.75^{\pm 2.17}$	$68.89^{\pm 1.65}$
		H&S	$88.55^{\pm 0.93}$	$71.29^{\pm 7.58}$	$56.96^{\pm 4.66}$
		TRACES	$92.64^{\pm 1.30}$	$88.15^{\pm 2.26}$	$69.75^{\pm 2.16}$
		Dream2Discern	$\mathbf{95.91}^{\pm 0.35}$	$\mathbf{91.16}^{\pm 0.63}$	$\mathbf{72.30}^{\pm 0.54}$
StepShield	Llama3.2-3B-it	SAFE	$66.18^{\pm 0.83}$	$59.57^{\pm 0.53}$	$51.28^{\pm 1.33}$
		H&S	$71.21^{\pm 7.03}$	$58.18^{\pm 2.71}$	$51.57^{\pm 1.26}$
		TRACES	$76.56^{\pm 3.20}$	$62.04^{\pm 2.45}$	$50.44^{\pm 4.21}$
		Dream2Discern	$\mathbf{90.08}^{\pm 1.78}$	$\mathbf{82.25}^{\pm 4.20}$	$\mathbf{55.01}^{\pm 2.26}$
	Llama3.1-8B-it	SAFE	$82.82^{\pm 2.41}$	$64.20^{\pm 0.71}$	$53.61^{\pm 0.24}$
		H&S	$87.54^{\pm 1.81}$	$59.26^{\pm 1.22}$	$52.61^{\pm 1.33}$
		TRACES	$\mathbf{89.59}^{\pm 2.65}$	$77.78^{\pm 3.24}$	$\mathbf{56.01}^{\pm 0.72}$
		Dream2Discern	$86.78^{\pm 1.92}$	$\mathbf{79.48}^{\pm 2.94}$	$49.06^{\pm 2.67}$

Table 3: Ablation on the perturbation strategy.

Benchmark	Base Model	Pert. Strategy	AUROC↑	wACC↑	TWA↑
AFTRAJ-2K	Qwen3-4B	w/o Perturbation	95.01	**90.96**	71.75
		w/ Random Pert.	94.40	89.16	68.24
		w/ *Dream* Pert.	**95.31**	90.36	**72.28**
	Qwen3-8B	w/o Perturbation	93.65	90.06	71.04
		w/ Random Pert.	94.60	89.46	69.80
		w/ *Dream* Pert.	**96.21**	**91.57**	**73.70**
StepShield	Qwen3-4B	w/o Perturbation	89.78	84.72	53.10
		w/ Random Pert.	91.20	84.26	52.78
		w/ *Dream* Pert.	**92.91**	**90.28**	**57.48**
	Qwen3-8B	w/o Perturbation	93.82	87.96	57.32
		w/ Random Pert.	94.37	**89.81**	**57.72**
		w/ *Dream* Pert.	**94.59**	88.43	56.42

fective runtime auditing with a compact backbone. Together, these results demonstrate the effectiveness of *Dream2Discern*, which combines loss-guided variant construction with outcome-supervised representation learning for runtime failure prediction.

Generalization across model families. To assess whether our training strategy extends beyond the Qwen3 family, we further compare *Dream2Discern* with SAFE, H&S, and TRACES using the same Llama3 backbones. Table 2 shows that *Dream2Discern* achieves the highest wACC on both benchmarks at both model sizes and leads all three reported metrics on AFTRAJ-2K. These results support the applicability of our training strategy across model families for runtime failure prediction.

4.3 Ablation and Further Analysis

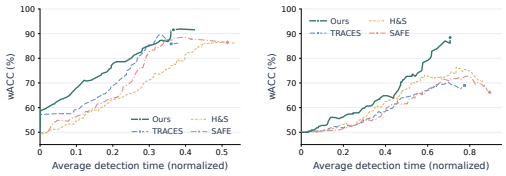
Ablation on the perturbation strategy. To assess the contribution of *Dream*, we compare *Dream2Discern* with the same auditor trained without perturbations or with random low-rank perturbations. The unperturbed variant trains the same LoRA adapters and prediction head using the prediction loss. The random variant retains the same perturbed operators, rank cap, and nominal radius, but replaces the loss-guided search with random sampling. As shown in Table 3, training with *Dream* consistently achieves the highest AUROC in all four settings. It also improves wACC and TWA over random perturbation training in most settings. These results support using prediction loss to construct hard latent variants that improve the auditor's discrimination between successful and failed executions. Section 4.4 further examines how the selected perturbations affect the auditor.

Detection accuracy–timeliness trade-off. We next examine whether *Dream2Discern* retains its predictive gains at operating points that favor earlier failure warnings, as a reliable runtime auditor should balance accuracy and promptness. Based on Qwen3-8B, we vary the nominal false-alarm level α from 0.01 to 1.00 to obtain various decision thresholds for *Dream2Discern*, SAFE, H&S, and TRACES. Figure 4 plots wACC against mean normalized detection time over all failed trajectories, with undetected failures assigned a value of one. On AFTRAJ-2K, *Dream2Discern* achieves higher wACC when early detection is prioritized, showing that its predictive gains are compatible with timely failure warnings. On StepShield, these gains extend across a broad range of operating points, supporting the effectiveness of *Dream2Discern* under different accuracy-timeliness requirements.

4.4 Mechanism Analysis

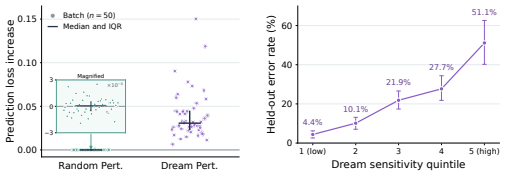
We analyze how *Dream* constructs hard latent variants and how sensitivity to its perturbations relates to held-out prediction errors. Appendix E.2 provides detailed experimental settings and analyses.

Variant construction. We first examine whether *Dream* constructs harder latent variants than random perturbations do. After training Qwen3-8B with *Dream2Discern*, we fix its learned parameters and sample 50 batches of 32 prefixes from the AFTRAJ-2K training set. For each batch, we construct a loss-guided *Dream* perturbation and sample random perturbations using the same settings. We temporarily apply each candidate and measure the change in batch BCE relative to the unperturbed auditor on the same prefixes. Each point in Figure 5a summarizes one batch, using the median of two draws for random perturbations. *Dream* increases loss on every batch, whereas random changes cluster near zero. This contrast shows that loss-guided search finds representation changes that make outcome prediction harder for the fixed auditor.



(a) AFTRAJ-2K (b) StepShield

Figure 4: Detection accuracy–timeliness trade-offs for Qwen3-8B. Markers denote $\alpha = 0.05$.



(a) Per-batch loss increase (b) Dream sensitivity

Figure 5: **(a)** Per-batch loss increases and **(b)** Dream sensitivity - Held-out error association.

Dream sensitivity. We next examine whether perturbations selected on training batches are informative about the auditor's prediction errors on held-out prefixes. We use a fixed Qwen3-4B auditor trained without perturbations and construct eight *Dream* perturbations from separate training batches. For each held-out AFTRAJ-2K prefix, we measure the absolute change in its failure score under each perturbation and average these changes to obtain its *Dream sensitivity*. We sort prefixes by sensitivity into five equal-sized groups and compute each group's error rate from the unperturbed predictions and recorded outcomes. Figure 5b shows that error rates increase from the lowest to the highest sensitivity group. This association suggests that loss-guided perturbations reflect weaknesses in the auditor's learned outcome discrimination, supporting the construction of hard latent variants around prediction difficulties that also arise on held-out prefixes.

4.5 CASE STUDY

Qualitative case studies. Figure 6 illustrates a held-out trajectory where *Dream2Discern* predicts the eventual failure at the ground-truth error step, two interactions before the environment reports `TEST_FAILED`. In this case, `CodeTester` mistakenly replaces `CodeWriter`'s correct `median` implementation with a `max` function during verification, thus leading the task to fail. As shown in the lower right panel, the auditor trained without perturbations first predicts failure at step 9, where the environment has already reported the test failure. However, when we temporarily apply *Dream* perturbations to the same auditor, its sensitivity exhibits a pronounced peak at step 7, as shown in the upper-right panel. This peak shows that the *Dream* perturbations elicit their strongest response at the held-out error step, which the unperturbed auditor has not yet flagged. The lower right panel further shows that after training on hard latent variants, *Dream2Discern* promptly predicts the eventual failure from the observed prefix at the ground-truth error step.

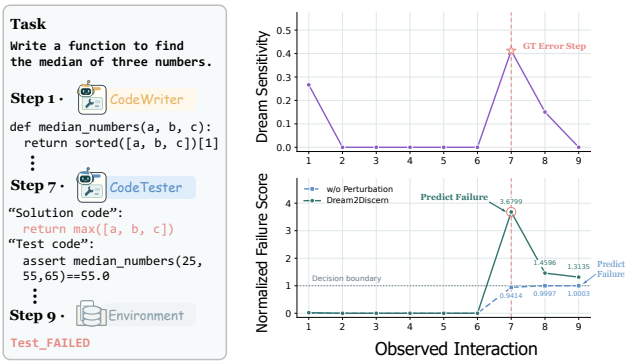


Figure 6: Case study of runtime failure prediction on a held-out coding trajectory from AFTRAJ-2K.

5 CONCLUSION

In this paper, we highlight that broader failure coverage alone does not identify the current auditor's hard cases, motivating implicit augmentation guided by prediction loss. To realize this idea, we present *Dream2Discern*, a runtime auditor that learns from hard latent variants without acquiring additional labeled trajectories. *Dream* constructs these variants through loss-guided operator perturbations, and *Discern* learns from them using the original outcome labels. The learned auditor operates without perturbations at runtime. Evaluations on AFTRAJ-2K and StepShield across model families demonstrate improved runtime failure prediction. These findings support targeting the auditor's current prediction difficulties to improve learning from a fixed trajectory corpus.

AI USE STATEMENT

The authors led the research, method development, experiments, and manuscript preparation. Generative AI tools were used in a supporting role to refine methodological and experimental choices, assist with mathematical formulation and proof development, support implementation and data analysis, and draft or edit portions of the manuscript. LLM use for trajectory editing, outcome checks, and baseline judgments is documented in Appendices E.1.1 and D.4. The authors critically reviewed and revised AI-assisted material against the implementation and recorded results, made the final scientific and editorial decisions, and take responsibility for the entire submission, including AI-assisted text, claims, code, and artifacts.

ETHICS STATEMENT

This work studies runtime failure prediction on existing agent-trajectory benchmarks to support reliable agent deployment. Auditor predictions can be incorrect and do not certify safe execution or prevent harmful actions. Deployment should retain task-specific safeguards and human oversight, and protect sensitive information in the execution histories supplied to the auditor.

REPRODUCIBILITY STATEMENT

The problem setting and method are specified in Sections 2 and 3, with the training algorithm and runtime calibration detailed in Appendix B. Appendix C provides the assumptions and complete proofs of the theoretical results. Dataset partitions, prefix construction, training configurations, baseline implementations, and evaluation metrics are described in Appendix D, and supplementary analysis protocols are given in Appendix E. An anonymized implementation with training and evaluation scripts and experiment configurations is included in the Supplementary Material.

REFERENCES

Avinash Baidya, Xinran Liang, Ruocheng Guo, Xiang Gao, and Kamalika Das. When evidence is sparse: Weakly supervised early failure alerting in dialogs and llm-agent trajectories. *arXiv preprint arXiv:2606.05414*, 2026.

Akash Bonagiri, Devang Borkar, Gerard Janno Anderias, Setareh Rafatirad, and Houman Homayoun. Causalflow: Causal attribution and counterfactual repair for llm agent failures. *arXiv preprint arXiv:2605.25338*, 2026.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *Advances in Neural Information Processing Systems*, 38, 2026.

Gloria Felicia, Zitha Sasindran, Jinfeng He, Michael Eniolade, Hemant Kumar, and Milan Hussain Angati. Stepshield: When, not whether to intervene on rogue agents. *arXiv preprint arXiv:2601.22136*, 2026.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Qiao Gu, Yuanliang Ju, Shengxiang Sun, Igor Gilitschenski, Haruki Nishimura, Masha Itkina, and Florian Shkurti. Safe: Multitask failure detection for vision-language-action models. *Advances in Neural Information Processing Systems*, 38:40041–40076, 2026.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a

multi-agent collaborative framework. In *The twelfth international conference on learning representations*, 2023.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Xinmiao Huang, Jinwei Hu, Rajarshi Roy, Changshun Wu, Yi Dong, and Xiaowei Huang. Prefixguard: From llm-agent traces to online failure-warning monitors. *arXiv preprint arXiv:2605.06455*, 2026.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, et al. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*, 2023.

Fanqi Kong, Ruijie Zhang, Huaxiao Yin, Guibin Zhang, Xiaofei Zhang, Ziang Chen, Zhaowei Zhang, Xiaoyuan Zhang, Song-Chun Zhu, and Xue Feng. Aegis: Automated error generation and attribution for multi-agent systems. In *International Conference on Learning Representations*, volume 2026, pp. 52994–53028, 2026.

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

Jiaqian Li, Yanshu Li, Boxuan Zhang, Ruixiang Tang, and Kuan-Hao Huang. Traces: Proactive safety auditing for multi-turn llm agents via trajectory-state modeling. *arXiv preprint arXiv:2605.27690*, 2026.

Dongrui Liu, Qihan Ren, Chen Qian, Shuai Shao, Yuejin Xie, Yu Li, Zhonghao Yang, Haoyu Luo, Peng Wang, Qingyu Liu, et al. Agentdog: A diagnostic guardrail framework for ai agent safety and security. *arXiv preprint arXiv:2601.18491*, 2026.

Andrey Malinin and Mark Gales. Uncertainty estimation in autoregressive structured prediction. *arXiv preprint arXiv:2002.07650*, 2020.

Changdae Oh, Seongheon Park, To Eun Kim, Jiatong Li, Wendi Li, Samuel Yeh, Sean Du, Hamed Hassani, Paul Bogdan, Dawn Song, et al. Uncertainty quantification in llm agents: Foundations, emerging challenges, and opportunities. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16219–16250, 2026.

Seongheon Park, Wendi Li, Changdae Oh, Samuel Yeh, Zsolt Kira, Michael Hagenow, and Sharon Li. Hide-and-seek in trajectories: Discovering failure signals for vla runtime monitoring. *arXiv preprint arXiv:2605.30834*, 2026.

Chen Qian, Peng Wang, Dongrui Liu, Junyao Yang, Dadi Guo, Ling Tang, Jilin Mei, Qihan Ren, Shuai Shao, Yong Liu, et al. The why behind the action: Unveiling internal drivers via agentic attribution. *arXiv preprint arXiv:2601.15075*, 2026.

Jie Ren, Jiaming Luo, Yao Zhao, Kundan Krishna, Mohammad Saleh, Balaji Lakshminarayanan, and Peter J Liu. Out-of-distribution detection and selective generation for conditional language models. *arXiv preprint arXiv:2209.15558*, 2022.

Ralf Römer, Adrian Kobras, Luca Worbis, and Angela Schoellig. Failure prediction at runtime for generative robot policies. *Advances in Neural Information Processing Systems*, 38:7631–7670, 2026.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris Maddison, and Tatsunori Hashimoto. Identifying the risks of lm agents with an lm-emulated sandbox. In *International Conference on Learning Representations*, volume 2024, pp. 27031–27098, 2024.

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

Alexander Timans, Christoph-Nikolas Straehle, Kaspar Sakmann, Christian A Naesseth, and Eric Nalisnick. Max-rank: Efficient multiple testing for conformal prediction. *arXiv preprint arXiv:2311.10900*, 2023.

Qizhou Wang, Junjie Ye, Feng Liu, Quanyu Dai, Marcus Kalander, Tongliang Liu, Jianye Hao, and Bo Han. Out-of-distribution detection with implicit outlier transformation. *arXiv preprint arXiv:2303.05033*, 2023.

Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. *Advances in neural information processing systems*, 33:2958–2969, 2020.

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

Jingxu Xie, Dylan Xu, Xuandong Zhao, and Dawn Song. Agentsynth: Scalable task generation for generalist computer-use agents. In *International Conference on Learning Representations*, volume 2026, pp. 155149–155177, 2026.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *International Conference on Learning Representations*, volume 2025, pp. 79822–79843, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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*, 2022.

Samuel Yeh, Yiwen Zhu, Shaleen Deep, and Sharon Li. Tracing agentic failure from the flow of success. *arXiv preprint arXiv:2607.12747*, 2026.

Boxuan Zhang, Jianing Zhu, Zeru Shi, Dongfang Liu, and Ruixiang Tang. Agentforesight: Online auditing for early failure prediction in multi-agent systems. *arXiv preprint arXiv:2605.08715*, 2026.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the llm agentic systems? *arXiv preprint arXiv:2509.03312*, 2025a.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*, 2025b.

Zhijie Zheng, Yu Li, Chen Qian, Yuqian Fu, Yanwei Fu, Lu Sheng, Jing Shao, and Dongrui Liu. Stepguard: Learning step-level guardrails with scalable supervision and safety-utility balancing. *arXiv preprint arXiv:2608.24777*, 2026.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, et al. Where llm agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*, 2025.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*, 2024.

A Detailed Related Work

LLM-based agentic systems. LLM-based agents combine reasoning, actions, and tool use to interact with external environments (Yao et al., 2022; Schick et al., 2023). Frameworks such as AutoGen (Wu et al., 2024), MetaGPT (Hong et al., 2023), and CAMEL (Li et al., 2023) organize these capabilities through specialized roles and inter-agent communication. GPTSwarm (Zhuge et al., 2024) further optimizes prompts and agent connectivity, while Magentic-One (Fourney et al., 2024) coordinates web browsing, file navigation, and code execution. Alongside these advances, reliable deployment requires assessing whether an ongoing execution is likely to fail, motivating our focus on runtime prediction from observed prefixes.

Failure analysis for LLM agents. Failure analysis characterizes errors in agent executions and provides guidance for diagnosis and improvement. MAST (Cemri et al., 2026) organizes multi-agent failures around system design, inter-agent misalignment, and task verification. Who&When (Zhang et al., 2025b) benchmarks attribution to responsible agents and decisive error steps, while AgenTracer (Zhang et al., 2025a) uses counterfactual replay and fault injection to construct supervision for a trained failure tracer. AgentDebug (Zhu et al., 2025) diagnoses errors across agent components and generates corrective feedback to support recovery. OAT (Yeh et al., 2026) learns trajectory dynamics from successful executions and uses deviations from these dynamics to locate error steps in failed trajectories. Extending attribution beyond failed executions, Qian et al. (2026) study which historical interactions and evidence drive an agent's actions. These studies support diagnosis and improvement; runtime prediction complements them by assessing terminal outcomes before execution finishes, when intervention remains possible.

Runtime failure prediction. Runtime auditors estimate outcomes from partial execution histories to support timely decisions. For vision-language-action policies, SAFE (Gu et al., 2026) predicts failure from internal features, while Hide-and-Seek (Park et al., 2026) discovers temporally structured failure signals through contrastive learning with trajectory-level labels. For LLM agents, AgentForesight (Zhang et al., 2026) trains an online auditor with decisive-error annotations, whereas TRACES (Li et al., 2026) models the temporal evolution of latent risk features from trajectory-level supervision. PrefixGuard (Huang et al., 2026) learns prefix-risk monitors from structured trace representations, and Baidya et al. (2026) combine sparse failure evidence with a learned stopping policy. Related safety guardrails include AgentDoG (Liu et al., 2026), which provides contextual risk diagnosis for agent trajectories. StepGuard (Zheng et al., 2026) combines context-matched safe and unsafe trajectories with reinforcement learning that balances performance across these classes to assess tool actions before execution. We focus on the complementary question of how to target the auditor's current prediction difficulties without expanding its trajectory corpus. *Dream2Discern* uses its prediction loss to construct hard latent variants of recorded prefixes and learns from them with the original outcome labels, without additional trajectory collection or verification.

B Additional Method Details

B.1 Training Algorithm and Solver Details

Algorithm 1 specifies the alternating updates after the supervised initialization in Appendix D.3. It uses the batch loss under the effective-operator perturbations in Section 3.2 and the scaled perturbation norm in Equation 3. Here $\mathcal{P}_r$ denotes truncated SVD retaining at most r leading singular components, and Eval computes a loss without gradients through the same numerical path for the unperturbed auditor and each candidate.

Solver details. Each batch starts from $V = 0$ and uses one normalized gradient step followed by SVD truncation and a global radial projection. Normalization uses the full scaled gradient norm with the numerical floor in Algorithm 1, rather than the retained norm g_r in Proposition 1. For the experimental settings in Appendix D.3, the resulting candidate satisfies, in exact arithmetic,

$$\overline{V}_{\mathcal{B},j} = \gamma\omega_j \frac{C_{j,r}}{d}, \qquad \|\overline{V}_{\mathcal{B}}\|_\omega = \gamma \frac{g_r}{d} \le \gamma,$$

where $C_{j,r}$ uses the singular components retained by the algorithm. Thus the candidate follows a maximizing direction of the linearized loss, but can have a smaller radius than the nominal linear

Algorithm 1 Dream2Discern: learning from hard latent variants

Require: Training-prefix index $\mathcal{D}$; initialized auditor $\Theta = (a, \psi)$; frozen backbone ϕ_0; suffix query/value projections $\mathcal{S}$; radius γ; rank cap r; iterations K; batch size $2m$.

Fix $\omega_j \leftarrow \|W_j^0\|_F$ for $j \in \mathcal{S}$; $\epsilon_{\text{num}} \leftarrow 10^{-12}$.
for $k = 1, \ldots, K$ **do**
  Form $\mathcal{B}$ by drawing m prefixes per class from $\mathcal{D}$ with replacement: parent group $\rightarrow$ trajectory $\rightarrow$ indexed prefix, uniformly at each level.
  Dream: hold Θ fixed and select a shared perturbation.
  $G_j \leftarrow \nabla_{V_j} \mathcal{L}_{\mathcal{B}}(\Theta, V)\big|_{V=0}$; $C_j \leftarrow \omega_j G_j$ for all $j \in \mathcal{S}$.
  $d \leftarrow \max\left\{ \left(\sum_{j \in \mathcal{S}} \|C_j\|_F^2 \right)^{1/2}, \epsilon_{\text{num}} \right\}$.
  $\overline{V}_{\mathcal{B},j} \leftarrow \mathcal{P}_r(\gamma \omega_j C_j / d)$ for all $j \in \mathcal{S}$.
  $\overline{V}_{\mathcal{B}} \leftarrow \overline{V}_{\mathcal{B}} \min\left\{ 1, \dfrac{\gamma}{\max\{\|\overline{V}_{\mathcal{B}}\|_\omega, \epsilon_{\text{num}}\}} \right\}$; $\widehat{V}_{\mathcal{B}} \leftarrow 0$.
  $\ell_0 \leftarrow \text{Eval}(\mathcal{L}_{\mathcal{B}}(\Theta, 0))$.
  for $\lambda \in (1, 2^{-1}, \ldots, 2^{-5})$ in order **do**
    if $\text{Eval}(\mathcal{L}_{\mathcal{B}}(\Theta, \lambda \overline{V}_{\mathcal{B}})) > \ell_0$ **then**
      $\widehat{V}_{\mathcal{B}} \leftarrow \lambda \overline{V}_{\mathcal{B}}$; **break**.
    end if
  end for
  Discern: learn from the returned variants.
  $g_\Theta \leftarrow \nabla_\Theta \mathcal{L}_{\mathcal{B}}(\Theta, \text{sg}[\widehat{V}_{\mathcal{B}}])$.
  Remove the temporary perturbation; $\Theta \leftarrow \text{AdamW}(\Theta, \text{clip}_1(g_\Theta))$.
end for
return Auditor f_Θ with $V = 0$.

solution; it is zero when $g_r = 0$. The subsequent acceptance test selects the first of six tested points that raises the measured batch loss, returning zero if all are rejected.

Discern update. The outer backward pass holds $\widehat{V}_{\mathcal{B}}$ constant and computes gradients only for the persistent adapters and prediction head. We remove the temporary perturbation, clip the global gradient norm at 1, and apply AdamW; optimizer settings are given in Appendix D.3. A zero returned perturbation yields the unperturbed batch update. The saved auditor contains the learned adapters and head and uses $V = 0$ at runtime.

Perturbation space. We perturb representation operators to couple the latent changes of different prefixes through a shared transformation. This follows the model-wise perspective of adversarial weight perturbation (Wu et al., 2020), which selects a weight perturbation using the loss over a batch. In our setting, Equation 5 gives the direct additive change $H_{q,j}^V V_j^\top$ at projection j for fixed incoming states. The shared operator couples these changes across tokens and prefixes, while their coefficients depend on the incoming states and their local output directions are restricted by the rank cap. This defines a structured search space for hard latent variants: query and value perturbations change the attention computation under a fixed prediction head.

B.2 Runtime Threshold Calibration

Threshold-fitting data. For *Dream2Discern*, we sample 40% of the parent groups represented in the training-prefix index using seed 0 and retain their successful trajectories with at least one nonterminal prefix. These trajectories come from the auditor's training partition and are distinct from the evaluation trajectories. With the trained auditor fixed at $V = 0$, we score every nonterminal prefix, indexed by $t = 1, \ldots, T_i - 1$, for each of the n_{cal} selected trajectories.

Stepwise reference distributions. Let $\mathcal{C}_t$ be the multiset of calibration scores at step t, with $m_t = |\mathcal{C}_t|$. Because every nonterminal prefix is scored, m_t is nonincreasing. For $n_{\text{cal}} \geq 20$, define $t_\star = \max\{t : m_t \geq 20\}$ and $b(t) = \min\{t, t_\star\}$, carrying the last supported reference distribution to

later steps. For sorted scores $s^{\mathrm{cal}}_{j,(1)} \le \cdots \le s^{\mathrm{cal}}_{j,(m_j)}$ in $\mathcal{C}_j$, define

$$F_t(x) = \frac{1}{m_{b(t)}} \sum_{s \in \mathcal{C}_{b(t)}} \mathbf{1}\{s \le x\},$$
$$Q_t^{\uparrow}(u) = s^{\mathrm{cal}}_{b(t),(1+\lceil (m_{b(t)}-1)u \rceil)}, \qquad u \in [0,1]. \tag{11}$$

The CDF includes ties, and $Q_t^{\uparrow}$ uses the `higher` quantile convention.

Trajectory maxima and thresholds. We summarize each trajectory by its maximum empirical percentile over nonterminal steps, following the rank-aggregation idea of max-rank (Timans et al., 2023). For nominal level $\alpha \in (0,1)$, define

$$R_i = \max_{1 \le t < T_i} F_t(s_{i,t}), \qquad k = \left\lceil (n_{\mathrm{cal}}+1)(1-\alpha) \right\rceil,$$
$$\zeta_t = \begin{cases} Q_t^{\uparrow}\big(R_{(k)}\big), & k \le n_{\mathrm{cal}}, \\ +\infty, & k > n_{\mathrm{cal}}, \end{cases} \tag{12}$$

where $R_{(k)}$ is the k-th smallest trajectory maximum. All nonterminal scores enter R_i, including those after $t_\star$ evaluated against the carried reference distribution. The resulting thresholds remain fixed at runtime, with $\zeta_t = \zeta_{t_\star}$ for $t > t_\star$ and failure predicted when $s_{i,t} \ge \zeta_t$ as in Equation 10. We use $\alpha = 0.05$ for the main *Dream2Discern* results; at the $\alpha = 1$ endpoint of the threshold sweep, we set $\zeta_t = Q_t^{\uparrow}(0)$. The same training trajectories determine both F_t and R_i, so α is a nominal parameter of this empirical calibration procedure, not a finite-sample false-alarm guarantee.

C Theoretical Analysis

C.1 Proof of Proposition 1

Proof. For any $V \in \mathcal{U}_{\gamma,r}$, set $U_j = V_j/\omega_j$. Since $\omega_j > 0$, this rescaling preserves rank and gives

$$\sum_{j \in \mathcal{S}} \|U_j\|_F^2 \le \gamma^2, \qquad \sum_{j \in \mathcal{S}} \langle G_j, V_j \rangle = \sum_{j \in \mathcal{S}} \langle C_j, U_j \rangle.$$

Order the singular values nonincreasingly and pad them with zeros when necessary. Since $\mathrm{rank}(U_j) \le r$, von Neumann's trace inequality followed by Cauchy–Schwarz yields

$$\begin{aligned} \langle C_j, U_j \rangle &\le \sum_{k=1}^{r} \sigma_k(C_j)\, \sigma_k(U_j) \\ &\le \left(\sum_{k=1}^{r} \sigma_k(C_j)^2 \right)^{1/2} \left(\sum_{k=1}^{r} \sigma_k(U_j)^2 \right)^{1/2} = \|C_{j,r}\|_F \, \|U_j\|_F. \end{aligned}$$

Summing over projections and applying Cauchy–Schwarz once more gives

$$\begin{aligned} \sum_{j \in \mathcal{S}} \langle C_j, U_j \rangle &\le \sum_{j \in \mathcal{S}} \|C_{j,r}\|_F \, \|U_j\|_F \\ &\le \left(\sum_{j \in \mathcal{S}} \|C_{j,r}\|_F^2 \right)^{1/2} \left(\sum_{j \in \mathcal{S}} \|U_j\|_F^2 \right)^{1/2} \le \gamma g_r. \end{aligned}$$

For $g_r > 0$, choose $U_j = \gamma C_{j,r}/g_r$. These matrices have rank at most r and satisfy $\sum_j \|U_j\|_F^2 = \gamma^2$. The truncated SVD also gives $\langle C_j, C_{j,r} \rangle = \|C_{j,r}\|_F^2$, so

$$\sum_{j \in \mathcal{S}} \langle C_j, U_j \rangle = \frac{\gamma}{g_r} \sum_{j \in \mathcal{S}} \|C_{j,r}\|_F^2 = \gamma g_r.$$

Thus, mapping back to $V_j = \omega_j U_j$ gives the feasible maximizer in Equation 6, which attains the upper bound. If $g_r = 0$, the same bound is zero and is attained at $V = 0$. □

C.2 Proof of Proposition 2

Proof. Fix Θ and $\mathcal{B}$ as in the proposition. Its remainder assumption states that, for every V on the segment from 0 to $\overline{V}_{\mathcal{B}}$,

$$\left| \mathcal{L}_{\mathcal{B}}(\Theta, V) - \mathcal{L}_{\mathcal{B}}(\Theta, 0) - \sum_{j \in \mathcal{S}} \langle G_j, V_j \rangle \right| \leq \frac{\beta}{2} \|V\|_{\omega}^2.$$

If $g_r = 0$, every retained gradient block is zero, so the candidate and returned perturbation are zero; Equation 9 holds with both sides zero. For $g_r > 0$, use the singular components retained by the algorithm to define $C_{j,r}$. This choice does not change g_r. Before radial rescaling, the truncated blocks in Algorithm 1 are $\gamma \omega_j C_{j,r}/d$, where $d > 0$ is its normalization denominator. Radial rescaling and the accepted step fraction apply a common nonnegative factor to these blocks. Since $(\omega_j C_{j,r})_{j \in \mathcal{S}}$ has scaled norm g_r, the returned point has the form

$$\widehat{V}_{\mathcal{B},j} = \rho_{\mathcal{B}}\, \omega_j \frac{C_{j,r}}{g_r}, \qquad j \in \mathcal{S},$$

including $\rho_{\mathcal{B}} = 0$ when the candidate is rejected. The truncated SVD satisfies $\langle C_j, C_{j,r} \rangle = \|C_{j,r}\|_F^2$. Together with $C_j = \omega_j G_j$, this gives

$$\begin{aligned} \sum_{j \in \mathcal{S}} \langle G_j, \widehat{V}_{\mathcal{B},j} \rangle &= \frac{\rho_{\mathcal{B}}}{g_r} \sum_{j \in \mathcal{S}} \langle C_j, C_{j,r} \rangle \\ &= \frac{\rho_{\mathcal{B}}}{g_r} \sum_{j \in \mathcal{S}} \|C_{j,r}\|_F^2 = \rho_{\mathcal{B}} g_r. \end{aligned}$$

By Equation 7, $\widehat{V}_{\mathcal{B}} = \lambda_{\mathcal{B}} \overline{V}_{\mathcal{B}}$ with $\lambda_{\mathcal{B}} \in [0, 1]$, so the returned point lies on the required segment. The remainder bound and the inner-product identity then yield Equation 9. □

A sufficient condition for accepting the full candidate. For a nonzero candidate, $g_r > 0$, and the same inner-product identity and remainder bound give

$$\mathcal{L}_{\mathcal{B}}(\Theta, \overline{V}_{\mathcal{B}}) - \mathcal{L}_{\mathcal{B}}(\Theta, 0) \geq \|\overline{V}_{\mathcal{B}}\|_{\omega} g_r - \frac{\beta}{2} \|\overline{V}_{\mathcal{B}}\|_{\omega}^2.$$

Consequently, $\beta \|\overline{V}_{\mathcal{B}}\|_{\omega} < 2 g_r$ ensures a strictly positive loss increase, so the first tested point is accepted in exact arithmetic. This condition also covers $\beta = 0$. If all six tested points are rejected, the search returns zero; a smaller untested step may still increase the loss.

D Additional Experimental Setups

D.1 Datasets and Evaluation Partitions

Dataset scope and labels. AFTraj-2K (Zhang et al., 2026) contains successful and failed multi-agent executions across mathematical reasoning, coding, and tool-based tasks. We use its recorded outcomes, with task failure mapped to $Y_i = 1$ and success to $Y_i = 0$. For StepShield (Felicia et al., 2026), we use the paired clean/rogue trajectory subset in our evaluation corpus, with policy violations (failure) mapped to $Y_i = 1$ and clean (success) trajectories to $Y_i = 0$. Table 4 distinguishes the available training trajectories from those retained after constructing the training-prefix index.

Partitions and prefix access. We retain the recorded AFTraj-2K train/test assignment. Grouping trajectories by the source-task stems of their identifiers yields 19 parent groups shared across the partitions, covering 26 test trajectories. For the StepShield subset, the split contains 639 training pairs and 108 test pairs, with both members of a pair assigned to the same partition. Training prefixes are selected using the annotations and balancing procedure in Appendix D.2. The training index includes terminal prefixes, whereas evaluation uses all nonterminal prefixes $1 \leq t < T_i$. Auditor inputs contain the task context and observed interactions but exclude outcome labels and decisive-step annotations; AFTraj-2K inputs also omit the stored thought field.

Table 4: Dataset statistics. Training-index rows count prefixes; test counts refer to trajectories.

	AFTRAJ-2K	StepShield
Available training trajectories	1,944	1,278
Trajectories in the training index	1,821	1,253
Training-index rows	8,454	7,828
Test trajectories	332	216
Test trajectories with $Y_i = 1$	163	108
Test trajectories with $Y_i = 0$	169	108

D.2 TRAINING PREFIX CONSTRUCTION

For the annotated training corpora, we select prefixes offline from completed trajectories. For a failed trajectory i, eligible prefixes end at $t = K_i, \ldots, T_i$, where K_i is its annotated decisive step. For a successful trajectory, we sample a starting step from the empirical distribution of training failure annotations, resampling starts at or beyond T_i and using a valid fallback when necessary. Candidate prefixes extend from this start through T_i, and each retains its trajectory's terminal outcome as the prediction target. We subsample equal numbers of success and failure prefixes within each combination of ending step, offset from the selected start, and terminal status. This balancing applies to the stored prefix pool; Algorithm 1 subsequently draws class-balanced batches through the group, trajectory, and prefix hierarchy. The annotations guide selection only and are excluded from auditor inputs. Prefix selection precedes token-budget truncation.

D.3 AUDITOR TRAINING AND REPORTING

Parameterization and input length. For all four backbones, we use the auditor parameterization in Section 3.2. Query and value projections after the middle Transformer block receive LoRA adapters of rank 8 with $\alpha_{\mathrm{LoRA}} = 16$, giving an adapter multiplier of $\alpha_{\mathrm{LoRA}}/8 = 2$. We use last-token readout by default: the final-layer hidden state of the last non-padding token is fed to a two-layer MLP with a 256-dimensional hidden layer and GELU activation. The input budget is 2,048 tokens; longer serialized prefixes retain the first 256 and last 1,792 tokens. Table 5 lists the optimization settings. We fix one perturbation radius per backbone across both benchmarks, taking the one whose *Dream* step most reduces the auditor's signed margin on training batches, as listed in Table 6.

Training budget and checkpoint selection. For each benchmark and backbone, we first fit the adapters and prediction head for 100 supervised steps. We then apply *Dream* and *Discern* for 1,057 steps on AFTRAJ-2K or 979 steps on StepShield and evaluate the final checkpoint. With logical batches of 32 prefixes, these budgets correspond to approximately four times the size of the respective training-prefix index. Because the hierarchical sampler draws with replacement, an epoch denotes this sampling budget, not a pass visiting every prefix exactly once. The threshold-fitting procedure and its data source are specified in Appendix B.2. Auditor training uses NVIDIA B200 GPUs with PyTorch 2.8.0 and CUDA 12.8.

Random seeds and uncertainty. Tables 1 and 2 report means and sample standard deviations over training seeds $0, 1, 2$ for *Dream2Discern*, SAFE, Hide-and-Seek, and TRACES across all four backbones. For *Dream2Discern*, runs within each benchmark and backbone share the same 100-step, seed-0 warm-start checkpoint and initialize a fresh optimizer and random state for subsequent training. The data split and threshold-fitting seed remain fixed across runs.

D.4 BASELINE IMPLEMENTATIONS

Open-weight judges. The Qwen and Llama instruction models are prompted to judge the observed prefix without additional task-specific fitting. Their failure scores are the `FAIL` probabilities from a softmax over the `FAIL` and `OK` answer logits. We use a 2,048-token prefix budget, preserving the task context and recent interactions when truncation is needed. LlamaGuard3-8B uses a task-failure prompt on AFTRAJ-2K and its standard safety taxonomy on StepShield. For each benchmark, we report the complete LlamaGuard variant winning more of the four test metrics among the native and task-specific prompts.

Table 5: Training settings for *Dream2Discern*.

Setting	Value
Optimizer	AdamW
Adapter and head learning rates	10^{-4}
Weight decay	0
Gradient clipping norm	1
Logical batch size	32 prefixes, 16 per outcome class
Microbatch size / accumulation steps	8 / 4
Supervised warm-start steps	100
Subsequent steps, AFTRAJ-2K / StepShield	1,057 / 979
Perturbation rank cap	$r = 8$
Gradient evaluations per *Dream* search	One at the unperturbed auditor
Candidate evaluation budget	At most six, as in Algorithm 1
Nominal threshold-fitting level	$\alpha = 0.05$

Table 6: Backbone-specific perturbation radii. Each value is used on both benchmarks.

Auditor backbone	γ
Qwen3-4B	3×10^{-4}
Qwen3-8B	3×10^{-3}
Llama-3.2-3B-Instruct	3×10^{-3}
Llama-3.1-8B-Instruct	3×10^{-4}

Closed-source judges. The closed-source comparisons use the recorded outputs of GPT-5.4, Claude-Sonnet-4.6, Grok-4.3, and Kimi-K2.7-Code. For these models, the returned continuous risk score is used for AUROC and FPR95, while a returned failure label supplies the binary prediction used for wACC and TWA. These decisions therefore use the models' reported labels rather than the fitted threshold band used for the other rows. Unparseable responses are retained as no-alarm predictions and assigned scores below all valid responses in that run. Each closed-source row represents one recorded evaluation run. The output budget is 4,096 tokens. On AFTRAJ-2K, GPT-5.4 also participated in planning fault injections during dataset construction.

AgentDebug (Zhu et al., 2025) and AgentForesight (Zhang et al., 2026). We adapt AgentDebug to Llama-3.1-8B-Instruct and apply its two-stage diagnosis to the observed prefix, removing the prompt's premise that failure has already occurred. An additional scoring stage converts the diagnosis into a failure score using the `FAIL` probability from a softmax over the `FAIL`/`OK` logits. Its prompt allocates 1,536 tokens to context and 3,072 tokens to the interaction block. For AgentForesight, we use the released AgentForesight-7B checkpoint with its native prompt and greedy decoding, without additional training in this study. At the generated verdict position, probabilities are normalized over `SAFE` and the ten digit tokens; the total digit probability supplies the alarm score. We use a 2,048-token budget, retaining task context and recent interactions.

Token uncertainty. We compute maximum and average token entropy and negative log-likelihood under Qwen3-8B. These scores measure a proxy model's uncertainty about the recorded text, since the logs do not provide the acting agents' own token probabilities. They do not use failure labels to fit their scoring functions; their operating thresholds are fitted separately.

Learned sequential scorers. SAFE (Gu et al., 2026), Hide-and-Seek (Park et al., 2026), and TRACES (Li et al., 2026) use frozen encoders from the same Qwen or Llama backbones as *Dream2Discern* in each comparison. The feature at each step is extracted from its observed prefix with a 2,048-token context budget, rather than from the isolated interaction alone. The downstream scorers are trained on the corresponding benchmark, using each method's objective and training unit. Our SAFE adaptation uses a one-layer LSTM with hidden width 256; the Hide-and-Seek adaptation uses a one-layer GRU of the same width with its inter- and intra-trajectory losses. SAFE uses 1,057 optimization steps on AFTRAJ-2K and 979 on StepShield, with 32 prefix rows per batch. Hide-and-Seek uses the same step counts with batches of 32 trajectory bags. TRACES uses 30 epochs of Stage-A pretraining, freezes that stage, and performs 528 Stage-B steps on AFTRAJ-2K or 489

on StepShield with batches of 64 prefix rows. The reported TRACES variant includes its auxiliary prefix and ranking losses, with loss weights $1 : 0.2 : 0.05$ for the final, prefix, and ranking objectives.

D.5 Evaluation Metrics

Continuous-score metrics. For each test trajectory, let

$$M_i = \max_{1 \le t < T_i} s_{i,t}. \tag{13}$$

We compute AUROC from M_i and Y_i. FPR95 is the fraction of successful trajectories exceeding or equaling the highest empirical score threshold that attains at least 95% recall on failed trajectories. These two metrics are computed from the continuous trajectory scores independently of the runtime threshold band.

Decisions and detection time. Let $\mathcal{P} = \{i : Y_i = 1\}$ and $\mathcal{N} = \{i : Y_i = 0\}$ denote the failed and successful test trajectories. For evaluation, a trajectory is counted as detected if any nonterminal prefix receives a failure prediction, and its detection time is the first such step:

$$D_i = \max_{1 \le t < T_i} \widehat{Y}_{i,t}, \qquad t_i^{\mathrm{det}} = \min\{t : 1 \le t < T_i,\ \widehat{Y}_{i,t} = 1\}. \tag{14}$$

The minimum is $+\infty$ when no prefix receives a failure prediction. This first-positive summary is used to compute evaluation metrics; the auditor itself continues to issue a current prediction at each prefix. Write $\mathrm{TP} = \{i \in \mathcal{P} : D_i = 1\}$ and $\mathrm{TN} = \{i \in \mathcal{N} : D_i = 0\}$. Then $\mathrm{TPR} = |\mathrm{TP}|/|\mathcal{P}|$ and $\mathrm{TNR} = |\mathrm{TN}|/|\mathcal{N}|$.

Accuracy and timeliness. Following the trajectory-level metrics used by Hide-and-Seek (Park et al., 2026), we compute

$$\begin{aligned} \mathrm{wACC} &= \pi_0\,\mathrm{TNR} + (1-\pi_0)\,\mathrm{TPR}, \qquad \pi_0 = \frac{|\mathcal{N}|}{|\mathcal{P}| + |\mathcal{N}|}, \\ \mathrm{TWA} &= \frac{1}{2}\left[\frac{1}{|\mathcal{P}|}\sum_{i \in \mathrm{TP}}\left(1 - \frac{t_i^{\mathrm{det}}}{T_i}\right) + \mathrm{TNR}\right]. \end{aligned} \tag{15}$$

With the empirical class proportions, wACC equals the fraction of correctly classified trajectories. TWA averages timeliness over all failed trajectories, so an undetected failure contributes zero. Detection steps are one-indexed and normalized by the full trajectory length T_i, even though terminal prefixes are excluded from evaluation. For the trade-off curves in Figure 4, we average $(t_i^{\mathrm{det}} - 1)/T_i$ over failed trajectories, assigning one to each undetected failure. Higher AUROC, wACC, and TWA are better; lower FPR95 is better. For score-based decisions we use nominal $\alpha = 0.05$; for the closed-source judges we use their returned labels, as described in Appendix D.4.

E Supplementary Experimental Analyses

E.1 Motivation Diagnostics

We detail the trajectory-editing, representation, and learning diagnostics in Section 3.1.

E.1.1 Trajectory Editing and Outcome Re-verification

We select 128 mathematical reasoning and coding trajectories from the training pool of AFTraj-2K and make four edit attempts at one nonterminal action per source. A GPT-5.4-mini editor uses the public task and observed history to propose an alternative action for the same agent, preserving the action type and tool schema without access to future interactions, outcome labels, or auditor feedback. After replaying preceding actions and matching their observations to the recorded history, we apply the edit and resume execution with GPT-5.4-mini under a 40-message limit covering replay and continuation. Coding outcomes are determined by running the last program submitted by `CodeWriter` against the source task's tests; these tests are not included in the continuation agents' system prompts. Mathematical outcomes are checked against reference answers with LLM-assisted semantic review. We retain 326 verified continuations from 97 sources with valid execution histories and resolved labels under consistent outcome checks. We compare each continuation's outcome with its verified source label to assess label reuse after editing and re-execution.

E.1.2 REPRESENTATION AND LEARNING DIAGNOSTICS

For Figure 2, we use a frozen Qwen3-4B encoder and the verified edits described in Appendix E.1.1. Panel (a) projects 97 original–edited prefix pairs onto jointly fitted principal components; density contours use all 326 edits with equal total weight per source trajectory. For panel (b), we train two-layer MLP prediction heads on original prefixes alone (*Original*) or the combined original and edited pool (+*Edited*), using the same initialization and 1,000 AdamW updates with batch size 128 and learning rate 10^{-3}. We expand the original training trajectories into all their prefixes and add prefixes from the verified continuations only in the +*Edited* condition. Feature standardization and class weights are computed from the original training pool and shared by both conditions. The plotted heads use seed 0. With the *Original* head fixed, we temporarily perturb the encoder's query and value projections (+*Dream*), using per-projection rank cap 8 and relative radius 3×10^{-4}; the perturbation is selected on a balanced batch of 32 original training prefixes. All conditions are evaluated on the same 2,820 held-out prefixes after excluding source groups represented in training, using the mean of the success-class and failure-class BCE losses. The loss plane varies the head parameters along the difference between the two trained heads and scales the selected encoder perturbation independently. The displayed coordinates are $x = v(L_D - L_O)$ and $y = u(L_E - L_O)$, where u interpolates the head parameters, v scales the perturbation, and L_O, L_E, L_D denote the three labeled losses. Arrow lengths reflect endpoint loss changes rather than parameter norms; dashed contours mark measurements beyond the perturbation budget.

Under the fixed training budget, adding the verified edits does not reduce held-out prediction loss. The training-selected *Dream* perturbation increases loss on the same held-out prefixes with the *Original* head fixed, supporting hard latent variant construction beyond the search batch.

E.2 MECHANISM ANALYSIS

Both experiments use AFTRAJ-2K with fixed seed-0 auditors, without updating their parameters during measurement. The loss comparison uses the final Qwen3-8B trained with *Dream2Discern*, whereas the sensitivity analysis uses the final Qwen3-4B trained without perturbations.

E.2.1 VARIANT CONSTRUCTION

Setup. We sample 50 batches of 32 prefixes from the 8,454 entries in the training-prefix index described in Appendix D.2. Using sampling seed 0, we draw with replacement uniformly over parent groups, trajectories within each group, and indexed prefixes within each trajectory. This diagnostic samples directly from the index without balancing the outcome classes. Both perturbation strategies act on the 34 adapted query/value projections with rank cap $r = 8$ and nominal relative radius $\gamma = 0.003$. The *Dream* candidate is the perturbation returned by the solver in Section 3.2. For each random draw, we form every projection perturbation as the product of two independent standard Gaussian matrices with inner dimension eight, then jointly rescale the projection perturbations to relative norm γ. We generate two independent random candidates per batch without selecting between them by prediction loss.

Measurement and aggregation. For each candidate V, we measure the change in mean batch BCE relative to the unperturbed auditor:

$$\Delta\mathcal{L}_{\mathcal{B}}(V) = \mathcal{L}_{\mathcal{B}}(\Theta, V) - \mathcal{L}_{\mathcal{B}}(\Theta, 0). \tag{16}$$

Both losses are recomputed on the same batch in evaluation mode with gradients disabled, and all prefixes receive equal loss weight. Each *Dream* point is the returned candidate's loss change, while each random point is the median loss change of its two draws. The summary marks in Figure 5a show the median and interquartile range, from the 25th to the 75th percentile, across the 50 points for each strategy. The magnified inset displays the same random points, including negative changes.

Results and interpretation. *Dream* increases loss on all 50 batches and exceeds the corresponding random summary on each batch. Its median loss increase is 0.0307, compared with 6.20×10^{-6} for random perturbations, whose changes occur on both sides of zero. The returned *Dream* perturbations have relative norms between 0.00261 and 0.00280, while random perturbations use the nominal boundary at 0.003. The comparison therefore evaluates the two construction strategies under the same operator scope, rank cap, and nominal budget. The contrast supports loss-guided

variant construction for a fixed auditor; the effect of learning from the resulting variants is assessed separately by the training ablation in Section 4.3.

E.2.2 DREAM SENSITIVITY

Setup. We hold the unperturbed Qwen3-4B fixed and construct eight *Dream* perturbations from independently sampled training batches, each containing 16 successful and 16 failed prefixes. Each solver call uses the same fixed auditor, the 34 adapted query/value projections, rank cap eight, and nominal relative radius 0.0003. We evaluate the returned candidates on nonterminal test prefixes after excluding every parent group represented in training and checking that no retained tokenized input exactly duplicates a training input after truncation. This leaves 2,820 prefixes from 306 trajectories and 198 parent groups; held-out labels are not used to construct perturbations.

Sensitivity and prediction error. For a held-out prefix $q = (i, t)$, let $s_q = f_\Theta(X_q)$ and let $s_q^{(k)}$ be the failure score obtained after applying the returned perturbation $\widehat{V}_{\mathcal{B}_k}$ to the same auditor. We define its *Dream sensitivity* as

$$H_q = \frac{1}{M} \sum_{k=1}^{M} \left| s_q^{(k)} - s_q \right|, \qquad M = 8. \tag{17}$$

Larger H_q means that the prefix's failure score responds more strongly to the *Dream* perturbations. Prediction errors are measured for the unperturbed auditor using $\widehat{Y}_q = \mathbf{1}\{s_q \geq \zeta_t\}$ and the recorded outcome Y_q. We fit the auditor's threshold band from its saved training predictions following the procedure in Appendix B.2, with calibration seed 0 and $\alpha = 0.05$, and keep this band fixed throughout the analysis.

Grouping and uncertainty. We divide prefixes into five groups using the 20th, 40th, 60th, and 80th percentiles of H_q, yielding 564 prefixes per group. To account for different numbers of prefixes per parent group, each prefix receives weight $w_q = 1/N_{g(q)}$, where $N_{g(q)}$ is its parent's prefix count in the entire held-out pool. For a sensitivity bin Q_j, the reported error rate is

$$\widehat{e}_j = \frac{\sum_{q \in Q_j} w_q \mathbf{1}\{\widehat{Y}_q \neq Y_q\}}{\sum_{q \in Q_j} w_q}. \tag{18}$$

These weights are fixed before binning and normalized within each bin. Error bars are 95% percentile intervals from 1,000 bootstrap resamples, each drawing 198 parent groups with replacement. All prefixes of a sampled parent inherit its resampling multiplicity, while the auditor, threshold band, perturbations, and bin boundaries remain fixed. The intervals quantify uncertainty over held-out parent groups conditional on this fixed auditor and perturbation bank.

Results and interpretation. The weighted error rate rises from 4.4% in the lowest sensitivity quintile to 51.1% in the highest, with increasing point estimates across all five bins. The perturbations use only training batches, so this association connects the loss-guided construction to prediction difficulties on held-out prefixes. These results support constructing hard latent variants that reflect weaknesses in the auditor's learned outcome discrimination beyond the construction batches.

E.3 ADDITIONAL TRAINING ABLATIONS

E.3.1 PERTURBATION STRATEGY

The three variants in Table 3 use training seed 0 and share the data, backbone, supervised initialization, logical batch size of 32, and optimizer-step budget. The unperturbed variant updates the adapters and prediction head on the original batch loss. For *Random Pert.*, each logical batch receives one shared random perturbation of the adapted query/value projections. We construct each projection perturbation as a product of two independent Gaussian matrices with inner dimension eight and jointly normalize them to $\|V\|_\omega = \gamma$, using the backbone-specific radii in Table 6. The random candidate is used directly, without loss-based selection or backtracking, and the adapters and head are updated on the perturbed batch loss. *Dream Pert.* uses Algorithm 1; all three variants are evaluated with $V = 0$. This training comparison is distinct from the fixed-auditor diagnostic in Appendix E.2.1.

E.3.2 PREFIX READOUT

To assess our default last-token readout, we compare it with mean pooling over all non-padding tokens in the final Transformer layer. For each readout, we train a separate auditor using the same backbone, data, perturbation settings, and optimization budget, including a 100-step supervised warm start with that readout. All runs use training seed 0. Table 7 shows that last-token readout yields higher AUROC in all four comparisons and higher wACC and TWA with Qwen3-4B. With Qwen3-8B, mean pooling yields higher TWA on both benchmarks and higher wACC on AFTRAJ-2K, while last-token readout retains higher wACC on StepShield. Last-token readout therefore improves score ranking in these comparisons, while its accuracy and timeliness gains at the calibrated operating point depend on the backbone and benchmark.

Table 7: Prefix-readout comparison with training seed 0. All metrics are percentages.

		AFTRAJ-2K			StepShield		
Backbone	Readout	AUROC↑	wACC↑	TWA↑	AUROC↑	wACC↑	TWA↑
Qwen3-4B	Last token (default)	95.31	90.36	72.28	92.91	90.28	57.48
	Mean pooling	91.44	89.76	69.56	87.27	79.17	47.61
Qwen3-8B	Last token (default)	96.21	91.57	73.70	94.59	88.43	56.42
	Mean pooling	92.95	92.17	74.56	91.98	87.04	57.06

E.4 SENSITIVITY TO PERTURBATION RADIUS

We vary $\gamma \in \{10^{-4}, 3\times10^{-4}, 10^{-3}, 3\times10^{-3}, 10^{-2}\}$ for Qwen3-4B and Qwen3-8B on both benchmarks, keeping the remaining training and evaluation settings fixed. All results use training seed 0. Table 8 shows that larger radii do not consistently improve either metric, and the best-performing setting differs across backbones and benchmarks. Relative to the main settings, $\gamma = 10^{-2}$ lowers TWA for both backbones on AFTRAJ-2K and substantially lowers wACC on StepShield.

Table 8: Perturbation-radius sensitivity with training seed 0. Metrics are percentages; † marks the radius used in the main experiments.

Backbone	γ	AFTRAJ-2K		StepShield	
		wACC↑	TWA↑	wACC↑	TWA↑
Qwen3-4B	10^{-4}	91.27	72.17	87.50	55.57
	$3\times10^{-4\dagger}$	90.36	72.28	90.28	57.48
	10^{-3}	91.27	73.58	86.11	53.72
	3×10^{-3}	91.87	73.45	80.56	54.42
	10^{-2}	89.76	67.66	78.70	55.82
Qwen3-8B	10^{-4}	90.06	71.79	90.74	57.50
	3×10^{-4}	89.46	71.83	87.50	55.69
	10^{-3}	91.57	74.07	89.81	57.04
	$3\times10^{-3\dagger}$	91.57	73.70	88.43	56.42
	10^{-2}	89.46	68.44	62.04	49.66

F LIMITATIONS

Our evaluation focuses on textual execution histories and binary terminal outcomes. Extending the auditor to visual observations or finer-grained outcome categories remains future work. We assess prediction quality on recorded trajectories, with downstream intervention left to the deployment system. Evaluating how auditor-guided replanning, tool restrictions, or human assistance affects task success and execution cost requires closed-loop experiments.