Hide-and-Seek in Trajectories: Discovering Failure Signals for VLA Runtime Monitoring

Seongheon Park[1] **Wendi Li**[1] **Changdae Oh**[1] **Samuel Yeh**[1]
Zsolt Kira[2] **Michael Hagenow**[1] **Sharon Li**[1]
[1]University of Wisconsin–Madison [2]Georgia Institute of Technology
{seongheon_park,sharonli}@cs.wisc.edu.

Abstract

Vision-Language-Action (VLA) models enable robots to follow natural language instructions and generalize across diverse tasks, but they remain vulnerable to execution failures that compromise reliability in real-world deployment. Detecting such failures during execution is therefore critical for the robust deployment of embodied systems. Existing failure detection methods either rely on expensive action resampling or external models, while alternatives propagate trajectory-level labels uniformly across every timestep, obscuring localized failure signals. In this paper, we propose **Hide-and-Seek**, a framework that formulates VLA failure detection as a coarsely supervised learning problem. By combining inter-trajectory and intra-trajectory contrastive objectives, Hide-and-Seek localizes failure-indicative actions and induces temporally structured failure signals from trajectory-level supervision alone, without any step-level annotation. We evaluate Hide-and-Seek on LIBERO, VLABench, and a real-world robotic platform across three representative VLA policies: OpenVLA, π_0, and $\pi_{0.5}$. Our method achieves state-of-the-art multi-task failure detection performance with a practical accuracy–timeliness trade-off under conformal prediction, and generalizes well to both seen and unseen tasks. Code and videos are available at our Project Page.

1 Introduction

Vision-language-action (VLA) models [1, 2, 3, 4, 5, 6, 7, 8] enable robot policies to follow natural language instructions and learn from heterogeneous, multimodal demonstrations, showing promising generalization to novel environments and embodiments [9]. Despite these advances, VLAs remain susceptible to diverse execution failures that undermine reliability and increase the cost of real-world deployment [10, 11]. Consider a household robot tasked with placing dishes into a cabinet: a subtle grasp failure, if undetected, can cascade into a dropped dish or costly recovery. Detecting such failures during execution is therefore critical for the robust deployment of embodied systems [12, 13, 14].

Existing failure detection methods fall short along two axes that jointly define the challenge. The first is *supervision cost*: obtaining step-level failure annotation is expensive and difficult to scale, requiring experts to identify the precise timestamps of error across long-horizon, stochastic trajectories. Recent work SAFE [15] sidesteps this issue by uniformly assigning trajectory-level failure labels to all timesteps, but this mislabels normal actions before failure onset, introducing substantial label noise that limits the efficacy of the learned detector. The second challenge is *computational practicality*: approaches based on action resampling [16, 17, 18] or external VLM judges [19, 20, 21] incur substantial inference overhead that precludes real-time deployment. Most prior methods are additionally designed for specialist policies on fixed tasks [12, 22, 23, 24], limiting their applicability to the diverse, open-world settings that modern generalist VLAs encounter. These limitations motivate a lightweight failure detector that localizes failure signals without step-level annotation.

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

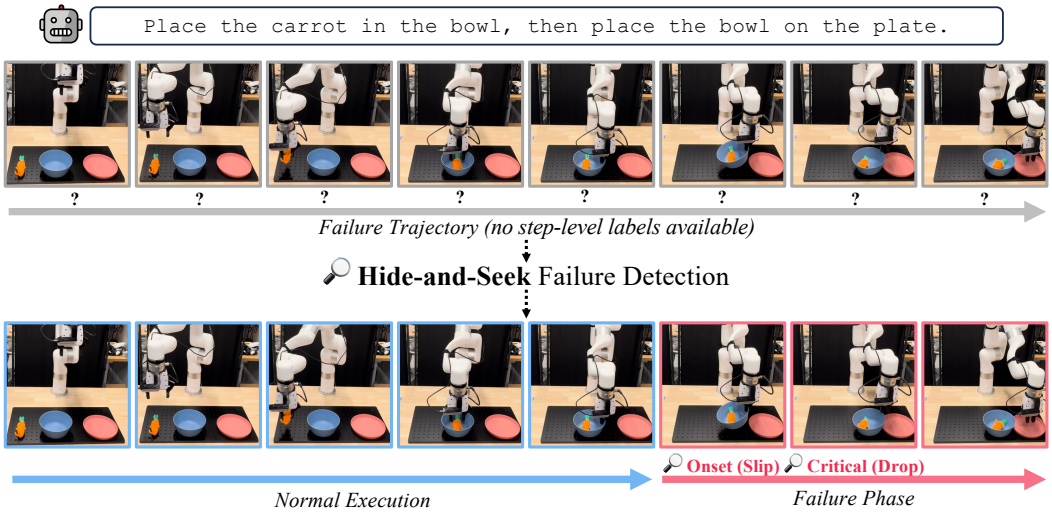


Figure 1: **Hide-and-Seek Failure Detection.** Failure trajectories contain a substantial amount of normal actions before failure onset, yet only a trajectory-level label is available during training, leaving the temporal structure entirely unknown (top). From this coarse supervision, Hide-and-Seek discovers the most failure-indicative actions (*e.g.*, the failure onset and subsequent critical event) by contrasting scores across and within trajectories, naturally separating normal execution from the failure phase without any step-level annotation (bottom).

To address this challenge, we propose **Hide-and-Seek**, a framework that discovers failure-indicative actions from trajectory-level supervision alone (Figure 1). We name our method to reflect the following metaphor: the failure signal is hidden among largely normal behavior within a trajectory, and the detector must seek it out using only the trajectory-level outcome as a clue. Rather than propagating the trajectory label uniformly to all time steps, Hide-and-Seek contrasts failure-indicative actions against non-failure actions at two complementary granularities. First, we propose an *inter-trajectory contrastive loss*, which enforces that the most failure-indicative step in a failure trajectory scores higher than the most failure-resembling step in a success trajectory. This guides the model to focus on discriminating the most salient failure signals. Moreover, we introduce an *intra-trajectory contrastive loss* that shapes score dynamics within each failure trajectory by encouraging the average failure score to be higher after the failure begins than before, converting trajectory-level supervision into a temporally structured failure signal without requiring any temporal annotation. Overall, the combination of the two losses imposes stronger structural regularity than uniform label propagation while adaptively discovering where the failure hides.

We evaluate Hide-and-Seek on two simulation benchmarks, LIBERO [25] and VLABench [26], as well as on a real-world robotic platform, using three representative VLA policies spanning both autoregressive and flow-matching-based paradigms: OpenVLA [1], π_0 [2], and $\pi_{0.5}$ [27]. Hide-and-Seek consistently outperforms all baselines on both seen and unseen tasks, surpassing the strongest classifier-based baseline by up to $+$**11.7**% bACC while maintaining a practical accuracy–timeliness trade-off. Notably, our method raises alarms near the annotated failure onset without any temporal supervision, and outperforms a VLM-based runtime monitor by $+$**13.1**% in accuracy while operating at over **2,000**$\times$ higher speed.

Our key contributions are summarized as follows:

- We propose Hide-and-Seek, a lightweight runtime failure detection framework that draws a novel connection between coarsely supervised learning and embodied failure detection, discovering failure-indicative actions from trajectory-level supervision alone.
- We design a contrastive objective that distinguishes failure-indicative actions from non-failure actions both across trajectories and within a trajectory, converting coarse trajectory-level supervision into a temporally structured failure signal.
- We comprehensively evaluate Hide-and-Seek on LIBERO and VLABench in simulation and on a real-world platform with OpenVLA, π_0, and $\pi_{0.5}$, achieving state-of-the-art performance on both seen and unseen tasks with a practical accuracy–timeliness trade-off.

2 Related Works

Vision-Language-Action models predict actions conditioned on multimodal input such as visual observations, robot proprioceptive states, and natural language instructions, and fall into two broad paradigms. The first paradigm treats action generation as an autoregressive sequence modeling problem [1, 28, 29, 30, 31, 32, 33], producing actions token-by-token in a manner analogous to language modeling [34, 35]. The second paradigm instead frames policy learning as a generative process over continuous action spaces, leveraging diffusion or flow-matching to produce actions through iterative denoising or transport [2, 27, 3, 4, 36, 37, 7]. Our method is architecture-agnostic and compatible with foundational VLA policies spanning both paradigms: the autoregressive model OpenVLA [1] and the flow-matching-based models π_0 [2] and $\pi_{0.5}$ [27]. This ensures the generality and broad applicability of our approach.

Failure detection in robot manipulation is critical for real-world deployment, as even minor errors can cascade into catastrophic outcomes [10, 11, 38]. Existing approaches fall into four categories: (1) OOD detection-based methods [39, 12, 40, 41, 42] treat successful executions as in-distribution and flag deviations as failures, yet struggle to generalize to unseen distributions. (2) VLM-based methods [19, 20, 21, 17] leverage vision-language models to identify and reason about failures from visual observations; however, they are limited by high inference latency and tend to detect failures only after they have fully manifested. (3) Multi-sampling-based methods [16, 17, 18] quantify uncertainty per action via ensemble sampling, but incur prohibitive inference costs that limit practical deployment. (4) Classifier-based methods [36, 24, 15, 23, 22] train failure classifiers either by collecting costly step-level annotations that are difficult to scale, or by propagating the trajectory label to every timestep, which introduces substantial label noise. This motivates a new classifier-based method that robustly learns a failure detector from trajectory-level labels alone.

Learning from coarse supervision is a label-efficient paradigm that leverages coarse-grained annotations to localize fine-grained signals [43, 44]. Often framed as weakly supervised learning, it has been successfully applied to object detection [45, 46], video temporal action localization [47, 48], and anomaly detection [49, 50]. For example, multiple instance learning [51] assigns a single label to a bag of instances and learns to identify the responsible instances from bag-level supervision alone. Despite the broad success across these domains, its connection to failure detection in embodied agents has not been explored. We draw this novel connection, enabling localized failure signal discovery from coarse trajectory-level supervision without per-step annotations.

3 Problem Setup

At each timestep t, a VLA model receives an observation o_t, consisting of RGB images, a natural language instruction, and the current robot state, and outputs an action a_t. We denote by $h_t \in \mathbb{R}^d$ the internal action embedding at time t, extracted from the model's action token or action head. To train and evaluate failure detectors, we execute the VLA across a set of tasks in simulation or the real world and collect rollout trajectories of embeddings, denoted as $\tau = (h_1, h_2, \ldots, h_T)$, where T is the episode horizon, and we write $\tau_{\le t} = (h_1, h_2, \ldots, h_t)$ for the prefix up to timestep t. Each trajectory is labeled with a trajectory-level outcome $y(\tau) \in \{0, 1\}$, where $y(\tau) = 1$ indicates failure and $y(\tau) = 0$ indicates success. Crucially, no step-level annotations are available, since obtaining them is costly and difficult to scale, particularly for long-horizon stochastic policies. We therefore address the *coarsely supervised failure detection* problem, a more practical alternative to the fully supervised version.

Definition 3.1 (Coarsely Supervised Failure Detection). *Given a set of successful trajectories $\mathcal{D}_s = \{\tau^{(k)} \mid y(\tau^{(k)}) = 0\}$ and a set of failure trajectories $\mathcal{D}_f = \{\tau^{(k)} \mid y(\tau^{(k)}) = 1\}$, where $y(\tau) \in \{0, 1\}$, the goal is to learn a fine-grained failure detector that discovers failure-indicative steps. At runtime, given a trajectory prefix $\tau_{\le t} = (h_1, h_2, \ldots, h_t)$, the detector produces a binary decision as follows:*

$$G(\tau_{\le t}) = \begin{cases} 1, & \text{if the trajectory prefix } \tau_{\le t} \text{ is predicted to result in task failure,} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

The central challenge is the mismatch between the granularity of supervision (trajectory-level) and the granularity of detection (a per-step decision). Our framework bridges this gap.

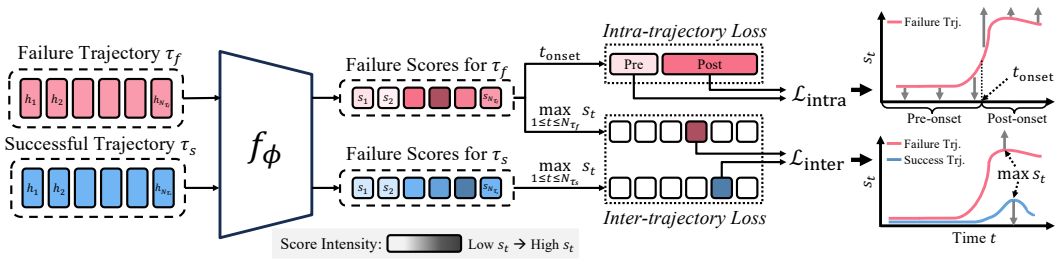


Figure 2: **Overall framework.** Given a failure-success trajectory pair (τ_f, τ_s), the detector f_ϕ produces per-step failure scores s_t for both failure and successful trajectories. $\mathcal{L}_{\text{inter}}$ (Eq. 2) enforces that the most failure-indicative action in the failure trajectory τ_f ranks higher than the hardest false-positive in the successful trajectory τ_s. $\mathcal{L}_{\text{intra}}$ (Eq. 3) defines a proxy failure onset as $t_{\text{onset}} = \arg\max_t(s_t - s_{t-1})$, and enforces temporal separation within τ_f by encouraging average post-onset scores to exceed average pre-onset scores. The two losses jointly shape the score dynamics so that s_t stays low during normal execution while sharply rising at t_{onset} in failure trajectories.

4 Hide-and-Seek Failure Detection

Overview. In this section, we introduce our framework **Hide-and-Seek** for failure detection. The name reflects the core challenge: the failure-inducing action is usually *hidden* among largely normal behavior, and the detector must *seek* it out from only trajectory-level supervision as a clue. For instance, in a task of placing a carrot in a bowl, the robot may successfully grasp the carrot, yet fail only at the placement stage. Naively assigning the trajectory-level failure label to every timestep treats all actions as equally failure-indicative, mislabeling many normal actions as failures, thereby introducing substantial label noise (see Section D.1).

Rather than propagating the trajectory label uniformly, our method contrasts failure-indicative actions against non-failure actions at two complementary granularities: (1) an *inter-trajectory contrastive loss* which contrasts the most salient failure-indicative action between failure and successful trajectories, (2) an *intra-trajectory contrastive loss* that encourages temporal separation in failure scores between pre-onset (normal execution) and post-onset (failure phase) within a failure trajectory, without requiring step-level annotations. The overall framework is illustrated in Figure 2.

4.1 Training Objective

We learn a sequential detector f_ϕ with a sigmoid output that maps each trajectory prefix to a scalar failure score[1]. Given a trajectory prefix $\tau_{\le t} = (h_1, h_2, \ldots, h_t)$ of VLA action embeddings, the detector produces $s_t = f_\phi(\tau_{\le t}) \in [0, 1]$. To optimize f_ϕ, we propose the following losses.

Inter-trajectory contrastive loss. Rather than classifying every step independently, we shift our focus to the most failure-indicative action step. The formulation of our loss rests on the key idea: for a failure trajectory $\tau_f \in \mathcal{D}_f$, at least one step carries a genuinely elevated failure score, and this score should be higher than the most failure-resembling action (the hardest false-positive, *e.g.*, a momentary hesitation or awkward intermediate pose that the robot successfully recovers from) in a successful trajectory $\tau_s \in \mathcal{D}_s$. This can be enforced with a margin-based contrastive loss:

$$\mathcal{L}_{\text{inter}} = \frac{1}{|\mathcal{D}_f|\,|\mathcal{D}_s|} \sum_{\tau_f \in \mathcal{D}_f} \sum_{\tau_s \in \mathcal{D}_s} \Big[\max\Big(0,\ m_r - \max_{1\le t\le N_{\tau_f}} s_t + \max_{1\le t\le N_{\tau_s}} s_t\Big)\Big], \tag{2}$$

where N_{τ_f} and N_{τ_s} are the lengths of τ_f and τ_s respectively, and $m_r > 0$ is a margin hyperparameter. The loss is computed over failure-success trajectory pairs (τ_f, τ_s) within each mini-batch. This objective makes no assumption about *where* the failure-indicative action occurs; instead, it adaptively discovers the most salient failure signal using a coarser form of supervision. At the same time, it enforces stronger structural regularity compared to propagating the trajectory label uniformly.

Intra-trajectory contrastive loss. Training with $\mathcal{L}_{\text{inter}}$ alone optimizes only a single peak per trajectory, leaving the surrounding temporal structure unsupervised. Since successful task completion

[1] In practice, f_ϕ can be instantiated as a lightweight sequence model, such as an LSTM, GRU, or Transformer. We use a single-layer LSTM as the default choice and provide additional ablations in Table 4 and Section D.3.

becomes less likely after failure onset [52], we introduce a second contrastive objective that enforces temporal separation around a proxy failure onset $t_{\text{onset}} = \arg\max_t(s_t - s_{t-1})$, defined as the point of the sharpest increase in failure score. Within each failure trajectory τ_f, we encourage the average post-onset score to exceed the average pre-onset score:

$$\mathcal{L}_{\text{intra}} = \frac{1}{|\mathcal{D}_f|} \sum_{\tau_f \in \mathcal{D}_f} \left[\max\left(0,\ m_o - \frac{1}{N_{\tau_f} - t_{\text{onset}} + 1} \sum_{t \geq t_{\text{onset}}} s_t + \frac{1}{t_{\text{onset}} - 1} \sum_{t < t_{\text{onset}}} s_t \right) \right], \tag{3}$$

where $m_o > 0$ is a margin hyperparameter. In Section C.1, we show that t_{onset} closely approximates the annotated failure onset and achieves comparable performance when used as the oracle reference, confirming that the sharpest score transition aligns with failure onset.

Overall objective. The complete training objective is a weighted combination of the inter-trajectory contrastive loss and the intra-trajectory contrastive loss:

$$\mathcal{L} = \mathcal{L}_{\text{inter}} + \lambda\, \mathcal{L}_{\text{intra}}, \tag{4}$$

where λ controls the loss weight. The objective overall aligns with the hide-and-seek intuition: $\mathcal{L}_{\text{inter}}$ seeks the hidden failure signal across trajectories, while $\mathcal{L}_{\text{intra}}$ sharpens the temporal boundary between normal execution and failure. Our method thus converts coarse trajectory-level supervision into a temporally structured failure signal without per-step annotation.

4.2 Runtime Monitoring via Conformal Prediction

After training, we deploy the failure detector to monitor VLA execution at inference time. At each timestep t, the detector produces a failure score $s_t = f_\phi(\tau_{\leq t}) \in [0, 1]$ and raises a failure alarm whenever s_t exceeds a threshold ζ_t. This realizes the binary decision $G(\tau_{\leq t})$ as a score-threshold rule:

$$G(\tau_{\leq t}) = \begin{cases} 1, & \text{if } s_t \geq \zeta_t \quad \text{(failure declared)}, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

A key challenge in deployment is selecting ζ_t appropriately: a fixed threshold fails to account for the natural temporal evolution of failure scores, while a poorly calibrated threshold leads to either excessive false alarms or missed detections. To address this, we adopt functional conformal prediction (CP) [53], which constructs a time-varying prediction band from a held-out calibration set of successful trajectories. Under the exchangeability assumption and a user-specified significance level $\alpha \in (0, 1)$, the band guarantees that for any new successful rollout, the trajectory-level false alarm rate is bounded by α. We employ the one-sided CP formulation [12, 15] where the threshold at each timestep is given by $\zeta_t = \mu_t + b_t$, with a time-varying mean μ_t and bandwidth b_t calibrated on the empirical score distribution of C successful calibration trajectories. During deployment, a failure is declared at the earliest timestep t for which $s_t \geq \zeta_t$, which we denote as the detection time.

5 Experiments

5.1 Setup

Implementation details. We extract per-timestep action embeddings from the action token or action head of the VLA. For autoregressive policies such as OpenVLA, we average the embeddings across layers and degrees of freedom to obtain a compact representation $h_t \in \mathbb{R}^d$. For flow-matching-based policies (e.g., π_0 and $\pi_{0.5}$), we extract hidden states before the velocity prediction head at the final denoising step, and align action chunks to obtain comparable per-timestep representations. To mitigate temporal redundancy [54], we apply non-overlapping sliding window aggregation with window size w, using average pooling within each window to produce $N_\tau = \lceil T/w \rceil$ window-level embeddings. Further implementation details are provided in Section B.

Benchmarks and models. We evaluate *multi-task* failure detection on two simulation benchmarks, LIBERO-10 [25] and VLABench [26], and a real-world UFactory xArm 6 platform, using representative VLA policies including OpenVLA [1], π_0 [2], and $\pi_{0.5}$ [27]. We adopt a held-out task protocol, reserving a subset of tasks as unseen to evaluate generalization. For all experiments, we report results averaged over 3 random seeds. Details on benchmarks and task splits are provided in Section E.

Table 1: Comparison of failure detection performance across policies on LIBERO-10. We report balanced accuracy (**bACC**), weighted accuracy (**wACC**), and time-weighted accuracy (**TWA**), averaged over 3 random seeds with standard deviation. **Bold** and underline denote the best and second-best results, respectively.

Policy: OpenVLA (Success Rate: 51.0%)

	Method		Seen Tasks			Unseen Tasks		
			bACC↑	wACC↑	TWA↑	bACC↑	wACC↑	TWA↑
Token Unc.	Max Entropy	*ICLR'21*	$0.514^{\pm 0.065}$	$0.527^{\pm 0.063}$	$0.455^{\pm 0.030}$	$0.521^{\pm 0.031}$	$0.518^{\pm 0.072}$	$0.488^{\pm 0.033}$
	Avg. Entropy	*ICLR'21*	$0.513^{\pm 0.020}$	$0.524^{\pm 0.038}$	$0.478^{\pm 0.035}$	$0.491^{\pm 0.059}$	$0.490^{\pm 0.049}$	$0.456^{\pm 0.032}$
	Max NLL	*ICLR'23*	$0.494^{\pm 0.054}$	$0.493^{\pm 0.039}$	$0.456^{\pm 0.050}$	$0.525^{\pm 0.033}$	$0.528^{\pm 0.057}$	$0.488^{\pm 0.022}$
	Avg. NLL	*ICLR'23*	$0.506^{\pm 0.039}$	$0.522^{\pm 0.028}$	$0.475^{\pm 0.024}$	$0.495^{\pm 0.027}$	$0.487^{\pm 0.045}$	$0.470^{\pm 0.022}$
OOD Det.	Mahalanobis	*ICML'18*	$0.670^{\pm 0.038}$	$0.667^{\pm 0.033}$	$0.604^{\pm 0.039}$	$0.513^{\pm 0.064}$	$0.495^{\pm 0.112}$	$0.504^{\pm 0.044}$
	Cosine k-NN	*ICML'22*	$0.758^{\pm 0.060}$	$0.757^{\pm 0.057}$	$0.632^{\pm 0.048}$	$0.639^{\pm 0.080}$	$0.590^{\pm 0.130}$	$0.522^{\pm 0.056}$
	PCA KMeans	*CoRL'24*	$0.534^{\pm 0.031}$	$0.544^{\pm 0.018}$	$0.494^{\pm 0.024}$	$0.486^{\pm 0.019}$	$0.494^{\pm 0.097}$	$0.442^{\pm 0.045}$
	RND	*ICML'24*	$0.526^{\pm 0.039}$	$0.523^{\pm 0.034}$	$0.493^{\pm 0.036}$	$0.509^{\pm 0.051}$	$0.504^{\pm 0.078}$	$0.478^{\pm 0.040}$
	LogpZO	*RSS'25*	$0.658^{\pm 0.091}$	$0.665^{\pm 0.089}$	$0.594^{\pm 0.067}$	$0.545^{\pm 0.028}$	$0.502^{\pm 0.049}$	$0.505^{\pm 0.030}$
Multi.	Cluster Entropy	*ICLR'23*	$0.552^{\pm 0.030}$	$0.524^{\pm 0.012}$	$0.516^{\pm 0.023}$	$0.570^{\pm 0.064}$	$0.578^{\pm 0.104}$	$0.548^{\pm 0.034}$
	EigenScore	*ICLR'24*	$0.663^{\pm 0.034}$	$0.636^{\pm 0.039}$	$0.536^{\pm 0.032}$	$0.671^{\pm 0.044}$	$0.672^{\pm 0.035}$	$0.524^{\pm 0.029}$
	STAC†	*CoRL'24*	$0.665^{\pm 0.057}$	$0.651^{\pm 0.073}$	$0.545^{\pm 0.051}$	$0.624^{\pm 0.062}$	$0.642^{\pm 0.089}$	$0.528^{\pm 0.042}$
	ACE	*NeurIPS'25*	$0.607^{\pm 0.071}$	$0.631^{\pm 0.083}$	$0.516^{\pm 0.041}$	$0.604^{\pm 0.087}$	$0.602^{\pm 0.072}$	$0.542^{\pm 0.033}$
Class.	SAFE-LSTM	*NeurIPS'25*	$0.670^{\pm 0.100}$	$0.665^{\pm 0.104}$	$0.593^{\pm 0.039}$	$0.686^{\pm 0.133}$	$0.666^{\pm 0.152}$	$0.553^{\pm 0.108}$
	SAFE-MLP	*NeurIPS'25*	$\underline{0.823}^{\pm 0.036}$	$\underline{0.819}^{\pm 0.034}$	$\underline{0.630}^{\pm 0.056}$	$\underline{0.775}^{\pm 0.062}$	$\underline{0.741}^{\pm 0.082}$	$\underline{0.590}^{\pm 0.039}$
	Ours		$\mathbf{0.852}^{\pm 0.051}$	$\mathbf{0.853}^{\pm 0.052}$	$\mathbf{0.660}^{\pm 0.035}$	$\mathbf{0.834}^{\pm 0.036}$	$\mathbf{0.828}^{\pm 0.034}$	$\mathbf{0.663}^{\pm 0.010}$

Policy: π_0 (Success Rate: 84.2%)

	Method		Seen Tasks			Unseen Tasks		
			bACC↑	wACC↑	TWA↑	bACC↑	wACC↑	TWA↑
OOD Det.	Mahalanobis	*ICML'18*	$0.788^{\pm 0.043}$	$0.840^{\pm 0.083}$	$\mathbf{0.715}^{\pm 0.043}$	$0.634^{\pm 0.054}$	$0.510^{\pm 0.057}$	$0.543^{\pm 0.037}$
	Cosine k-NN	*ICML'22*	$0.580^{\pm 0.040}$	$0.841^{\pm 0.031}$	$0.565^{\pm 0.032}$	$0.623^{\pm 0.035}$	$0.696^{\pm 0.100}$	$0.542^{\pm 0.025}$
	RND	*ICML'24*	$0.706^{\pm 0.067}$	$0.784^{\pm 0.131}$	$0.629^{\pm 0.069}$	$0.520^{\pm 0.031}$	$0.643^{\pm 0.052}$	$0.505^{\pm 0.021}$
	PCA KMeans	*CoRL'24*	$0.591^{\pm 0.140}$	$0.595^{\pm 0.234}$	$0.476^{\pm 0.048}$	$0.461^{\pm 0.093}$	$0.471^{\pm 0.122}$	$0.394^{\pm 0.089}$
	LogpZO	*RSS'25*	$0.786^{\pm 0.040}$	$0.835^{\pm 0.078}$	$0.691^{\pm 0.035}$	$0.681^{\pm 0.050}$	$0.679^{\pm 0.136}$	$0.574^{\pm 0.028}$
Multi.	Cluster Entropy	*ICLR'23*	$0.548^{\pm 0.096}$	$0.749^{\pm 0.163}$	$0.454^{\pm 0.047}$	$0.732^{\pm 0.158}$	$0.604^{\pm 0.201}$	$0.552^{\pm 0.088}$
	EigenScore	*ICLR'24*	$0.691^{\pm 0.057}$	$0.761^{\pm 0.037}$	$0.534^{\pm 0.030}$	$0.613^{\pm 0.143}$	$0.668^{\pm 0.100}$	$0.495^{\pm 0.071}$
	STAC	*CoRL'24*	$0.619^{\pm 0.068}$	$0.751^{\pm 0.057}$	$0.501^{\pm 0.044}$	$0.490^{\pm 0.044}$	$0.596^{\pm 0.093}$	$0.440^{\pm 0.033}$
	ACE	*NeurIPS'25*	$0.672^{\pm 0.074}$	$0.764^{\pm 0.099}$	$0.572^{\pm 0.049}$	$0.609^{\pm 0.066}$	$0.630^{\pm 0.063}$	$0.505^{\pm 0.029}$
Class.	SAFE-LSTM	*NeurIPS'25*	$0.774^{\pm 0.024}$	$0.851^{\pm 0.041}$	$0.663^{\pm 0.047}$	$0.609^{\pm 0.057}$	$0.528^{\pm 0.176}$	$0.524^{\pm 0.080}$
	SAFE-MLP	*NeurIPS'25*	$\underline{0.870}^{\pm 0.038}$	$\underline{0.915}^{\pm 0.047}$	$0.688^{\pm 0.060}$	$\underline{0.801}^{\pm 0.083}$	$\underline{0.867}^{\pm 0.091}$	$\underline{0.648}^{\pm 0.088}$
	Ours		$\mathbf{0.885}^{\pm 0.041}$	$\mathbf{0.926}^{\pm 0.054}$	$\underline{0.693}^{\pm 0.021}$	$\mathbf{0.892}^{\pm 0.011}$	$\mathbf{0.921}^{\pm 0.044}$	$\mathbf{0.705}^{\pm 0.043}$

†Adapted to OpenVLA's single-step output via MMD between adjacent timesteps (see Section G.2).

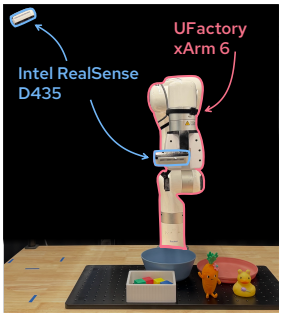
Evaluation metrics. We evaluate runtime failure detection using three complementary metrics, treating failure trajectories as positives and successful trajectories as negatives. (1) **Balanced Accuracy (bACC)** equally weights both classes, and serves as our primary class-balanced metric. (2) **Weighted Accuracy (wACC)** weights each class by its empirical frequency, reflecting deployment scenarios with class imbalance. (3) **Time-weighted Accuracy (TWA)** [16] jointly captures correctness and timeliness by penalizing late detections based on their detection time. All three metrics depend on the conformal threshold, set by a parameter α that controls the false alarm rate on successful trajectories. We report results averaged over $\alpha \in \{0.15, 0.20, 0.25\}$ across all experiments. Full definitions are provided in Section B.3.

Baselines. We comprehensively compare our approach against 12 action-based failure detection baselines: **(1) OOD detection-based methods** model the in-distribution embedding space, extended with failure embeddings to match our pairwise supervision: Mahalanobis [55], Cosine k-NN [56], PCA-KMeans [39], RND [42], and LogpZO [12]. (2) **Multi-sampling-based methods**[2] assess uncertainty through repeated action sampling: Cluster Entropy [57], EigenScore [58], STAC [17], and ACE [16]. (3) **Classifier-based methods** train a failure classifier on frozen VLA representations: SAFE [15]. For OpenVLA, we additionally evaluate (4) **Token uncertainty-based methods**, which estimate failure likelihood from per-action token probabilities: Entropy [59], Negative log-likelihood (NLL) [60]. Details about each baseline are provided in Section G.

Table 2: Comparison of failure detection performance with $\pi_{0.5}$ on VLABench.

	Method		Seen Tasks bACC↑	Seen Tasks wACC↑	Seen Tasks TWA↑	Unseen Tasks bACC↑	Unseen Tasks wACC↑	Unseen Tasks TWA↑
	Policy: $\pi_{0.5}$ (Success Rate: 40.3%)							
OOD Det.	Mahalanobis	*ICML'18*	$0.637^{\pm 0.076}$	$0.519^{\pm 0.114}$	$0.581^{\pm 0.079}$	$0.611^{\pm 0.101}$	$0.548^{\pm 0.092}$	$0.545^{\pm 0.073}$
	Cosine k-NN	*ICML'22*	$0.535^{\pm 0.023}$	$0.384^{\pm 0.044}$	$0.485^{\pm 0.035}$	$0.704^{\pm 0.072}$	$0.676^{\pm 0.095}$	$0.584^{\pm 0.088}$
	PCA KMeans	*CoRL'24*	$0.642^{\pm 0.052}$	$0.669^{\pm 0.085}$	$0.529^{\pm 0.002}$	$0.616^{\pm 0.065}$	$0.618^{\pm 0.068}$	$0.520^{\pm 0.053}$
	RND	*ICML'24*	$0.628^{\pm 0.007}$	$0.618^{\pm 0.012}$	$0.545^{\pm 0.013}$	$0.551^{\pm 0.044}$	$0.537^{\pm 0.014}$	$0.507^{\pm 0.017}$
	LogpZO	*RSS'25*	$0.618^{\pm 0.067}$	$0.503^{\pm 0.102}$	$0.552^{\pm 0.071}$	$0.674^{\pm 0.030}$	$0.628^{\pm 0.028}$	$0.581^{\pm 0.021}$
Multi.	Cluster Entropy	*ICLR'23*	$0.629^{\pm 0.050}$	$0.656^{\pm 0.069}$	$0.602^{\pm 0.029}$	$0.526^{\pm 0.055}$	$0.506^{\pm 0.067}$	$0.490^{\pm 0.050}$
	EigenScore	*ICLR'24*	$0.553^{\pm 0.027}$	$0.487^{\pm 0.030}$	$0.532^{\pm 0.024}$	$0.566^{\pm 0.017}$	$0.538^{\pm 0.025}$	$0.554^{\pm 0.017}$
	STAC	*CoRL'24*	$0.615^{\pm 0.058}$	$0.586^{\pm 0.072}$	$0.556^{\pm 0.068}$	$0.591^{\pm 0.017}$	$0.598^{\pm 0.048}$	$0.576^{\pm 0.021}$
	ACE	*NeurIPS'25*	$0.596^{\pm 0.032}$	$0.545^{\pm 0.008}$	$0.553^{\pm 0.033}$	$0.591^{\pm 0.018}$	$0.595^{\pm 0.019}$	$0.552^{\pm 0.006}$
Class.	SAFE-LSTM	*NeurIPS'25*	$0.629^{\pm 0.012}$	$0.524^{\pm 0.030}$	$0.587^{\pm 0.027}$	$0.484^{\pm 0.063}$	$0.533^{\pm 0.115}$	$0.482^{\pm 0.065}$
	SAFE-MLP	*NeurIPS'25*	$0.788^{\pm 0.111}$	$0.766^{\pm 0.106}$	$0.581^{\pm 0.084}$	$0.641^{\pm 0.029}$	$0.634^{\pm 0.084}$	$0.500^{\pm 0.038}$
	Ours		$\mathbf{0.856}^{\pm 0.075}$	$\mathbf{0.827}^{\pm 0.098}$	$\mathbf{0.662}^{\pm 0.100}$	$\mathbf{0.713}^{\pm 0.035}$	$\mathbf{0.709}^{\pm 0.023}$	$\mathbf{0.608}^{\pm 0.017}$

Table 3: Real-world robot setup (left) and failure detection performance with $\pi_{0.5}$ (right).



Method	CUBE (Success Rate: 61.2%) Seen bACC	Seen TWA	Unseen bACC	Unseen TWA	KITCHEN (Success Rate: 63.7%) Seen bACC	Seen TWA	Unseen bACC	Unseen TWA
Mahalanobis	0.683	0.652	0.681	0.623	0.779	0.754	0.630	0.608
Cosine k-NN	0.674	0.617	0.658	0.647	0.693	0.612	0.644	0.592
PCA KMeans	0.525	0.509	0.472	0.463	0.541	0.522	0.566	0.553
RND	0.517	0.505	0.547	0.538	0.679	0.661	0.615	0.605
LogpZO	0.699	0.666	0.660	0.639	0.895	0.811	0.837	0.794
SAFE-LSTM	0.723	0.597	0.714	0.623	0.846	0.780	0.595	0.577
SAFE-MLP	**0.992**	0.847	0.797	0.650	0.784	0.748	0.878	0.768
Ours	0.966	**0.852**	**0.914**	**0.800**	**0.968**	**0.863**	**0.972**	**0.876**

5.2 Main Experiments

Simulation environment. Tables 1 and 2 report failure detection performance on LIBERO-10 and VLABench across three VLA policies: OpenVLA, π_0 and $\pi_{0.5}$. Hide-and-Seek consistently achieves state-of-the-art performance, outperforming all baselines across models and benchmarks in terms of bACC, wACC, and TWA on both seen and unseen tasks. OOD detection-based methods perform competitively on seen tasks but degrade substantially on unseen tasks, *e.g.*, Mahalanobis drops from 67.0% to 51.3% bACC on OpenVLA, revealing their limitation in generalizing beyond the training distribution. Multi-sampling-based methods generally underperform classifier-based approaches, as uncertainty estimates derived from repeated action sampling do not reliably reflect task-level failure without access to any failure supervision. Among classifier-based methods, both SAFE-LSTM and SAFE-MLP train on trajectory-level labels, diluting the decision boundary by treating all timesteps uniformly. SAFE-MLP partially mitigates this with a linearly increasing temporal weight, but this fixed heuristic is agnostic to when failure actually manifests. Our method instead adaptively discovers the most failure-indicative moments from trajectory-level supervision alone, outperforming SAFE-MLP by $+$**6.8**% bACC and $+$**8.1**% TWA on seen tasks, and $+$**7.2**% bACC and $+$**10.8**% TWA on unseen tasks on VLABench with $\pi_{0.5}$. *Qualitative results are provided in Section A.2.*

Real-world robot. Table 3 reports failure detection performance on a UFactory xArm 6 platform with $\pi_{0.5}$ across two task suites, CUBE and KITCHEN. Hide-and-Seek consistently outperforms all baselines on both seen and unseen tasks, achieving the best TWA across all four settings and gaining $+$**11.7**% bACC and $+$**15.0**% TWA on unseen CUBE over SAFE-MLP. Baseline performance varies substantially across splits, *e.g.*, SAFE-MLP achieves 99.2% bACC on seen CUBE but drops to 79.7% on unseen. In contrast, Hide-and-Seek maintains strong and stable performance across all four settings, demonstrating that failure signals discovered from trajectory-level supervision transfer effectively to real-world execution and generalize beyond the simulator. Due to real-time deployment constraints, we exclude multi-sampling-based methods that incur significant inference overhead. Experimental details are provided in Section E.2.

[2]We use $N = 10$ samples per step to balance estimation quality and the sampling cost of large VLA models.

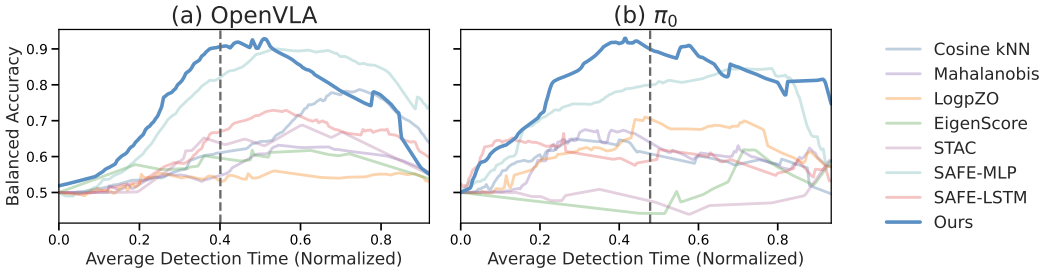


Figure 3: Detection accuracy–timeliness tradeoff on LIBERO-10 with OpenVLA and π_0. Gray dashed line indicates the average normalized failure onset time across failure trajectories.

Table 4: Architecture.

Method	OpenVLA	π_0
MLP	0.765	0.743
GRU	0.834	0.893
Transformer	0.804	0.824
LSTM	0.843	0.888

Table 5: Component analysis.

$\mathcal{L}_{\text{inter}}$	$\mathcal{L}_{\text{intra}}$	OpenVLA	π_0
✓		0.825	0.856
	✓	0.579	0.601
✓	✓	0.843	0.888

Table 6: Comparison with VLM-based runtime monitor.

Method	bACC ↑	Latency ↓
VLM Monitor	0.712	2.343s
Ours	0.843	0.001s

5.3 Ablation Studies

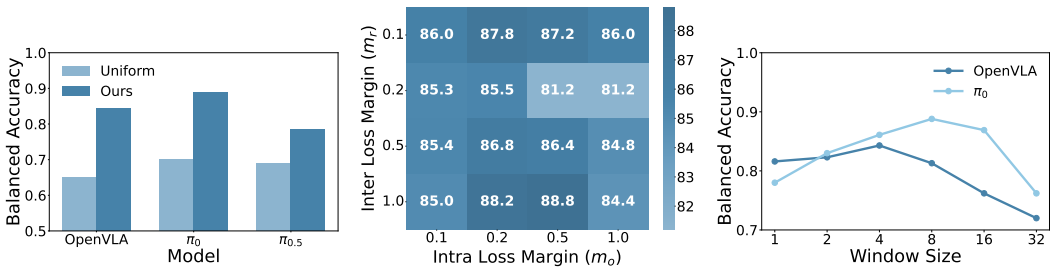
Detection accuracy–timeliness trade-off. A reliable failure detector must balance accuracy and promptness. In our framework, the user-specified α controls this trade-off: stricter α yields more conservative thresholds with delayed detection, while looser α enables earlier detection at the cost of more false alarms. We sweep $\alpha \in [0.01, 1.0]$ to trace a curve of operating points for each method on unseen LIBERO-10 splits in Figure 3. For each α, we compute the average detection time (normalized by trajectory length) and the corresponding bACC; curves closer to the top-left indicate better trade-off. For temporal reference, we annotate the failure onset of each failed rollout using GPT-5.2 [61] on recorded videos, shown as vertical dashed lines (see Section F.2 for human agreement). Hide-and-Seek achieves the best trade-off on LIBERO-10 with both OpenVLA and π_0, with curves consistently closer to the top-left than baselines, indicating earlier and more accurate failure detection. We further quantify detection timeliness at the fixed α used in deployment in Section D.2.

Detector architecture. We compare our LSTM-based [62] detector[3] against three alternatives in Table 4: an MLP, a GRU [63], and a single-layer Transformer [64]. All variants share the same input features and training setup, differing only in the backbone. Context-utilizing backbones (GRU, Transformer, LSTM) substantially outperform the per-step MLP, indicating that our formulation benefits from modeling temporal context when identifying failure-indicative actions. Among these, the LSTM and GRU outperform the Transformer. We conjecture that recurrent backbones offer a more suitable inductive bias for sequential failure detection under limited training data. We adopt the LSTM as our default backbone due to its consistently strong performance across settings.

Loss component analysis. We ablate the contribution of each training objective in Table 5. Training with $\mathcal{L}_{\text{inter}}$ alone already achieves strong performance, surpassing the best-performing baseline (SAFE-MLP in Table 1) by $+2.6\%$ on OpenVLA and $+2.1\%$ on π_0. This result indicates that inter-trajectory contrastive loss effectively extracts failure-indicative signals even under weak trajectory-level supervision. $\mathcal{L}_{\text{intra}}$ alone yields degraded performance, as it relies on the failure onset proxy, which is only meaningful once $\mathcal{L}_{\text{inter}}$ has shaped the score dynamics to reflect failure-indicative structure. Combining both objectives yields consistent improvements on both OpenVLA ($+1.8\%$) and π_0 ($+3.2\%$), demonstrating that $\mathcal{L}_{\text{intra}}$ effectively strengthens the onset signal that $\mathcal{L}_{\text{inter}}$ discovers implicitly, converting it into explicit supervision without requiring any additional annotation.

Hide-and-Seek is more efficient and effective than VLM-based runtime monitoring. We compare our method against a VLM-based runtime monitor that prompts Qwen3-VL-8B-Instruct [65] with online video question answering on historical frames up to the current timestep, outputting a discrete success/failure judgment per frame on LIBERO-10 with OpenVLA. A trajectory is classified

[3] Unless otherwise noted, the remaining ablations report bACC averaged over seen and unseen tasks.



(a) Uniform vs. ours. (b) Effect of loss margin m. (c) Effect of window size w.

Figure 4: (a) Comparison between uniform trajectory-level labeling and our approach; (b) effect of margins m_r and m_o on detection performance; (c) effect of window size w on detection performance. All results report balanced accuracy averaged over seen and unseen tasks on LIBERO-10.

as a failure if at least one frame is judged as a failure. As shown in Table 6, Hide-and-Seek outperforms the VLM-based monitor by a substantial margin ($+13.1\%$ bACC), while operating at over **2,000**$\times$ **higher speed** (0.001s vs. 2.343s per step on a single NVIDIA A6000 GPU). This demonstrates that VLA latent action embeddings provide a richer failure signal than visual observations alone and are considerably more practical for real-time deployment.

Comparison with uniform trajectory-level labeling. To validate the necessity of our formulation, we compare against a baseline that uniformly assigns the trajectory-level label to every timestep and trains with a margin loss on all steps under otherwise identical settings, including the window size. As shown in Figure 4a, our approach substantially outperforms uniform labeling [15], achieving gains of $+$**19.4**$\%$ bACC on OpenVLA and $+$**18.6**$\%$ on π_0. These results indicate that uniform supervision introduces label noise by treating correct actions as failure-indicative (see Section D.1) whereas our method isolates the critical failure signals necessary for reliable detection.

Effect of loss margin. We study the sensitivity of Hide-and-Seek to the margin hyperparameters m_r in $\mathcal{L}_{\text{inter}}$ and m_o in $\mathcal{L}_{\text{intra}}$ by sweeping both on LIBERO-10 with π_0. As shown in Figure 4b, performance remains stable with bACC varying by less than 4% from the mean across all configurations, indicating that Hide-and-Seek is robust to margin choices rather than requiring precise tuning. The best performance is achieved at $m_r = 1.0, m_o = 0.5$ (88.8% bACC), and we adopt this setting as our default for π_0. Insufficiently large margins ($m_r, m_o \leq 0.2$) provide weak separation between failure and successful actions, while an overly large intra loss margin ($m_o = 1.0$) over-constrains the intra-trajectory score structure.

Effect of window size. We ablate the window size w used for the sliding window aggregation in Section 5.1. As shown in Figure 4c, moderate window sizes consistently improve performance over small ones, confirming that temporal aggregation reduces noise in step-level representations while preserving informative dynamics. We observe that $w = 4$ performs best for OpenVLA, whereas $w = 8$ yields the highest accuracy for π_0. This trend suggests that appropriate windowing captures the local temporal continuity of robot actions. Notably, for π_0, the optimal window size aligns with its action prediction horizon (replanning interval), indicating that matching the aggregation scale to the policy's action chunking structure further enhances failure signal extraction.

Additional ablations on (1) failure onset proxies, (2) thresholding strategies, (3) layer-wise effects, and (4) inference-time policy steering are provided in Section C.

6 Conclusion

We present Hide-and-Seek, a lightweight runtime failure detector for VLA models that discovers failure-indicative actions from trajectory-level supervision alone. By formulating failure detection as a coarsely supervised learning problem, Hide-and-Seek combines an inter-trajectory contrastive objective to identify critical failure-indicative actions with an intra-trajectory objective that sharpens score separation around the inferred failure onset. Extensive experiments on LIBERO, VLABench, and a real-world robotic platform with OpenVLA, π_0, and $\pi_{0.5}$ show that Hide-and-Seek consistently outperforms existing methods on both seen and unseen tasks, achieving state-of-the-art multi-task failure detection accuracy with a practical accuracy–timeliness trade-off.

Acknowledgement

We gratefully acknowledge the anonymous reviewers for their constructive feedback, as well as Yu Wang, Shawn Im, Heecheol Kim, and Dahye Kim for their valuable feedback. Seongheon Park, Wendi Li, Changdae Oh, Samuel Yeh, and Sharon Li are supported in part by the AFOSR Young Investigator Program under award number FA9550-23-1-0184, the Office of Naval Research (ONR) under award number N000142612508, the National Science Foundation under awards IIS-2237037 and IIS-2331669, the Alfred P. Sloan Fellowship, Open Philanthropy (now Coefficient Giving), and Schmidt Sciences Foundation. Zsolt Kira was supported in part by the National Science Foundation under Grant No. IIS-2239292 and Schmidt Sciences Foundation. We would also like to thank Tutor Intelligence for donating the robots used in the hardware experiments.

References

[1] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. In *CoRL*, 2024.

[2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[3] Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, et al. $\pi^*_{0.6}$: a vla that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025.

[4] Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[5] Xiao-Ming Wu, Bin Fan, Kang Liao, Jian-Jian Jiang, Runze Yang, Yihang Luo, Zhonghua Wu, Wei-Shi Zheng, and Chen Change Loy. Vlanext: Recipes for building strong vla models. *arXiv preprint arXiv:2602.18532*, 2026.

[6] Jianke Zhang, Xiaoyu Chen, Yanjiang Guo, Yucheng Hu, and Jianyu Chen. VLM4VLA: Revisiting vision-language-models in vision-language-action models. In *ICLR*, 2026.

[7] Marcel Torne, Karl Pertsch, Homer Walke, Kyle Vedder, Suraj Nair, Brian Ichter, Allen Z Ren, Haohuan Wang, Jiaming Tang, Kyle Stachowicz, et al. Mem: Multi-scale embodied memory for vision language action models. *arXiv preprint arXiv:2603.03596*, 2026.

[8] Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. Memer: Scaling up memory for robot control via experience retrieval. In *ICLR*, 2026.

[9] Chao Xu, Suyu Zhang, Yang Liu, Baigui Sun, Weihong Chen, Bo Xu, Qi Liu, Juncheng Wang, Shujun Wang, Shan Luo, et al. An anatomy of vision-language-action models: From modules to milestones and challenges. *arXiv preprint arXiv:2512.11362*, 2025.

[10] Monica L Visinsky, Joseph R Cavallaro, and Ian D Walker. Robotic fault detection and fault tolerance: A survey. *Reliability Engineering & System Safety*, 1994.

[11] Quazi Marufur Rahman, Peter Corke, and Feras Dayoub. Run-time monitoring of machine learning for robotic perception: A survey of emerging trends. *IEEE Access*, 2021.

[12] Chen Xu, Tony Khuong Nguyen, Emma Dixon, Christopher Rodriguez, Patrick Miller, Robert Lee, Paarth Shah, Rares Ambrus, Haruki Nishimura, and Masha Itkina. Can we detect failures without failure data? uncertainty-aware runtime failure detection for imitation learning policies. In *RSS*, 2025.

[13] Yusuf Ali, Gryphon Patlin, Karthik Kothuri, Muhammad Zubair Irshad, Wuwei Liang, and Zsolt Kira. Eve: A generator-verifier system for generative policies. *arXiv preprint arXiv:2512.21430*, 2025.

[14] Ram Ramrakhya, Matthew Chang, Xavier Puig, Ruta Desai, Zsolt Kira, and Roozbeh Mottaghi. Grounding multimodal llms to embodied agents that ask for help with reinforcement learning. *arXiv preprint arXiv:2504.00907*, 2025.

[15] Qiao Gu, Yuanliang Ju, Shengxiang Sun, Igor Gilitschenski, Haruki Nishimura, Masha Itkina, and Florian Shkurti. Safe: Multitask failure detection for vision-language-action models. In *NeurIPS*, 2025.

[16] Ralf Römer, Adrian Kobras, Luca Worbis, and Angela P Schoellig. Failure prediction at runtime for generative robot policies. In *NeurIPS*, 2025.

[17] Christopher Agia, Rohan Sinha, Jingyun Yang, Zi-ang Cao, Rika Antonova, Marco Pavone, and Jeannette Bohg. Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress. In *CoRL*, 2024.

[18] Suhyeok Jang, Dongyoung Kim, Changyeon Kim, Youngsuk Kim, and Jinwoo Shin. Verifier-free test-time sampling for vision language action models. In *ICLR*, 2026.

[19] Jessie Yuan, Yilin Wu, and Andrea Bajcsy. When to act, ask, or learn: Uncertainty-aware policy steering. *arXiv preprint arXiv:2602.22474*, 2026.

[20] Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, and Yijie Guo. Aha: A vision-language-model for detecting and reasoning over failures in robotic manipulation. In *ICLR*, 2025.

[21] Carl Qi, Xiaojie Wang, Silong Yong, Stephen Sheng, Huitan Mao, Sriram Srinivasan, Manikantan Nambi, Amy Zhang, and Yesh Dattatreya. Self-refining vision language model for robotic failure detection and reasoning. In *ICLR*, 2026.

[22] Alec Farid, David Snyder, Allen Z Ren, and Anirudha Majumdar. Failure prediction with statistical guarantees for vision-based robot control. In *RSS*, 2022.

[23] Huihan Liu, Shivin Dass, Roberto Martín-Martín, and Yuke Zhu. Model-based runtime monitoring with interactive imitation learning. In *ICRA*, 2024.

[24] Junwon Seo, Kensuke Nakamura, and Andrea Bajcsy. Uncertainty-aware latent safety filters for avoiding out-of-distribution failures. In *CoRL*, 2025.

[25] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. In *NeurIPS*, 2023.

[26] Shiduo Zhang, Zhe Xu, Peiju Liu, Xiaopeng Yu, Yuan Li, Qinghui Gao, Zhaoye Fei, Zhangyue Yin, Zuxuan Wu, Yu-Gang Jiang, et al. Vlabench: A large-scale benchmark for language-conditioned robotics manipulation with long-horizon reasoning tasks. In *ICCV*, 2025.

[27] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

[28] Ruijie Zheng, Yongyuan Liang, Shuaiyi Huang, Jianfeng Gao, Hal Daumé III, Andrey Kolobov, Furong Huang, and Jianwei Yang. TraceVLA: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies. In *ICLR*, 2025.

[29] Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, et al. Spatialvla: Exploring spatial representations for visual-language-action model. In *RSS*, 2025.

[30] Jun Cen, Siteng Huang, Yuqian Yuan, Kehan Li, Hangjie Yuan, Chaohui Yu, Yuming Jiang, Jiayan Guo, Xin Li, Hao Luo, et al. Rynnvla-002: A unified vision-language-action and world model. *arXiv preprint arXiv:2511.17502*, 2025.

[31] Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

[32] Yicheng Liu, Shiduo Zhang, Zibin Dong, Baijun Ye, Tianyuan Yuan, Xiaopeng Yu, Linqi Yin, Chenhao Lu, Junhao Shi, Luca Jiang-Tao Yu, Liangtao Zheng, Jingjing Gong, Tao Jiang, Xipeng Qiu, and Hang Zhao. FASTer: Toward powerful and efficient autoregressive vision–language–action models with learnable action tokenizer and block-wise decoding. In *ICLR*, 2026.

[33] Yuqi Wang, Xinghang Li, Wenxuan Wang, Junbo Zhang, Yingyan Li, Yuntao Chen, Xinlong Wang, and Zhaoxiang Zhang. Unified vision-language-action model. In *ICLR*, 2026.

[34] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

[35] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023.

[36] Jose Barreiros, Andrew Beaulieu, Aditya Bhat, Rick Cory, Eric Cousineau, Hongkai Dai, Ching-Hsin Fang, Kunimatsu Hashimoto, Muhammad Zubair Irshad, Masha Itkina, et al. A careful examination of large behavior models for multitask dexterous manipulation. *arXiv preprint arXiv:2507.05331*, 2025.

[37] Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

[38] Pasquale Antonante, Heath G Nilsen, and Luca Carlone. Monitoring of perception systems: Deterministic, probabilistic, and learning-based fault detection and identification. *Artificial Intelligence*, 2023.

[39] Huihan Liu, Yu Zhang, Vaarij Betala, Evan Zhang, James Liu, Crystal Ding, and Yuke Zhu. Multi-task interactive robot fleet learning with visual world models. In *CoRL*, 2024.

[40] Shijie Zhou, Bin Zhu, Jiarui Yang, Xiangyu Zhao, Jingjing Chen, and Yu-Gang Jiang. Rc-nf: Robot-conditioned normalizing flow for real-time anomaly detection in robotic manipulation. In *CVPR*, 2026.

[41] Rohan Sinha, Amine Elhafsi, Christopher Agia, Matthew Foutter, Edward Schmerling, and Marco Pavone. Real-time anomaly detection and reactive planning with large language models. In *RSS*, 2024.

[42] Nantian He, Shaohui Li, Zhi Li, Yu LIU, and You He. Rediffuser: Reliable decision-making using a diffuser with confidence estimation. In *ICML*, 2024.

[43] Minqi Jiang, Chaochuan Hou, Ao Zheng, Xiyang Hu, Songqiao Han, Hailiang Huang, Xiangnan He, Philip S Yu, and Yue Zhao. Weakly supervised anomaly detection: A survey. *arXiv preprint arXiv:2302.04549*, 2023.

[44] Dingwen Zhang, Junwei Han, Gong Cheng, and Ming-Hsuan Yang. Weakly supervised object localization and detection: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[45] Peng Tang, Xinggang Wang, Xiang Bai, and Wenyu Liu. Multiple instance detection network with online instance classifier refinement. In *CVPR*, 2017.

[46] Fang Wan, Chang Liu, Wei Ke, Xiangyang Ji, Jianbin Jiao, and Qixiang Ye. C-mil: Continuation multiple instance learning for weakly supervised object detection. In *CVPR*, 2019.

[47] Pilhyeon Lee, Jinglu Wang, Yan Lu, and Hyeran Byun. Weakly-supervised temporal action localization by uncertainty modeling. In *AAAI*, 2021.

[48] Xiaojun Tang, Junsong Fan, Chuanchen Luo, Zhaoxiang Zhang, Man Zhang, and Zongyuan Yang. Ddg-net: Discriminability-driven graph network for weakly-supervised temporal action localization. In *ICCV*, 2023.

[49] Waqas Sultani, Chen Chen, and Mubarak Shah. Real-world anomaly detection in surveillance videos. In *CVPR*, 2018.

[50] Guansong Pang, Chunhua Shen, Huidong Jin, and Anton Van Den Hengel. Deep weakly-supervised anomaly detection. In *KDD*, 2023.

[51] Marc-André Carbonneau, Veronika Cheplygina, Eric Granger, and Ghyslain Gagnon. Multiple instance learning: A survey of problem characteristics and applications. *Pattern recognition*, 2018.

[52] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

[53] Jacopo Diquigiovanni, Matteo Fontana, and Simone Vantini. The importance of being a band: Finite-sample exact distribution-free prediction sets for functional data. *arXiv preprint arXiv:2102.06746*, 2021.

[54] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *RSS*, 2023.

[55] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *NeurIPS*, 2018.

[56] Yiyou Sun, Yifei Ming, Xiaojin Zhu, and Yixuan Li. Out-of-distribution detection with deep nearest neighbors. In *ICML*, 2022.

[57] Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *ICLR*, 2023.

[58] Chao Chen, Kai Liu, Ze Chen, Yi Gu, Yue Wu, Mingyuan Tao, Zhihang Fu, and Jieping Ye. Inside: Llms' internal states retain the power of hallucination detection. In *ICLR*, 2024.

[59] Andrey Malinin and Mark Gales. Uncertainty estimation in autoregressive structured prediction. In *ICLR*, 2021.

[60] 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. In *ICLR*, 2023.

[61] Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

[62] Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, 2012.

[63] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[65] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

[66] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[67] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

[68] Anastasios N Angelopoulos and Stephen Bates. Conformal prediction: A gentle introduction. *Foundations and Trends in Machine Learning*, 2023.

[69] Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Patel, Jalal Naghiyev, Yann LeCun, and Ravid Shwartz-Ziv. Layer by layer: Uncovering hidden representations in language models. In *ICML*, 2025.

[70] Zhangqi Jiang, Junkai Chen, Beier Zhu, Tingjin Luo, Yankun Shen, and Xu Yang. Devils in middle layers of large vision-language models: Interpreting, detecting and mitigating object hallucinations via attention lens. In *CVPR*, 2025.

[71] Raymond Tsao, Andrew Wagenmaker, and Sergey Levine. Learning process rewards via success visitation matching for efficient rl. *arXiv preprint arXiv:2606.23640*, 2026.

[72] Seongheon Park, Heecheol Kim, Shulin Tian, Lilika Makabe, Namiko Saito, Katsushi Ikeuchi, Sharon Li, and Yasuyuki Matsushita. Diver: Decision-critical verifier learning for vla test-time scaling. *arXiv preprint arXiv:2610.04933*, 2026.

[73] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 2008.

[74] 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. In *ICLR*, 2022.

[75] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *NeurIPS*, 2023.

[76] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM computing surveys*, 2023.

[77] Hanchao Liu, Wenyuan Xue, Yifei Chen, Dapeng Chen, Xiutian Zhao, Ke Wang, Liping Hou, Rongjun Li, and Wei Peng. A survey on hallucination in large vision-language models. *arXiv preprint arXiv:2402.00253*, 2024.

[78] Yiyang Zhou, Chenhang Cui, Jaehong Yoon, Linjun Zhang, Zhun Deng, Chelsea Finn, Mohit Bansal, and Huaxiu Yao. Analyzing and mitigating object hallucination in large vision-language models. In *ICLR*, 2024.

[79] Min-Hsuan Yeh, Max Kamachee, Seongheon Park, and Yixuan Li. Halluentity: Benchmarking and understanding entity-level hallucination detection. In *TMLR*, 2025.

[80] Seongheon Park, Changdae Oh, Hyeong Kyu Choi, Xuefeng Du, and Sharon Li. Vauq: Vision-aware uncertainty quantification for lvlm self-evaluation. In *ACL Findings*, 2026.

[81] Stephanie Lin, Jacob Hilton, and Owain Evans. Teaching models to express their uncertainty in words. In *TMLR*, 2022.

[82] Tobias Groot and Matias Valdenegro-Toro. Overconfidence is key: Verbalized uncertainty evaluation in large language and vision-language models. In *Proceedings of the 4th Workshop on Trustworthy Natural Language Processing (TrustNLP 2024)*, 2024.

[83] Ruiyang Zhang, Hu Zhang, and Zhedong Zheng. Vl-uncertainty: Detecting hallucination in large vision-language model via uncertainty estimation. *arXiv preprint arXiv:2411.11919*, 2024.

[84] Amos Azaria and Tom Mitchell. The internal state of an LLM knows when it's lying. In *EMNLP Findings*, 2023.

[85] Xuefeng Du, Chaowei Xiao, and Yixuan Li. Haloscope: Harnessing unlabeled llm generations for hallucination detection. In *NeurIPS*, 2024.

[86] Seongheon Park, Xuefeng Du, Min-Hsuan Yeh, Haobo Wang, and Yixuan Li. Steer llm latents for hallucination detection. In *ICML*, 2025.

[87] Mengjia Niu, Hamed Haddadi, and Guansong Pang. Robust hallucination detection in llms via adaptive token selection. In *NeurIPS*, 2025.

[88] Seongheon Park and Sharon Li. Glsim: Detecting object hallucinations in lvlms via global-local similarity. In *NeurIPS*, 2025.

[89] Kunal Menda, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Ensembledagger: A bayesian approach to safe imitation learning. In *IROS*, 2019.

[90] Michael Hagenow and Julie A Shah. Realm: Real-time estimates of assistance for learned models in human-robot interaction. *IEEE Robotics and Automation Letters*, 2025.

[91] Michael Hagenow, Emmanuel Senft, Robert Radwin, Michael Gleicher, Bilge Mutlu, and Michael Zinn. Corrective shared autonomy for addressing task variability. *IEEE Robotics and Automation Letters*, 2021.

[92] Jiho Lee, Hayun Lee, Jonghyeon Kim, Kyungjae Lee, and Eunwoo Kim. Self-corrective task planning by inverse prompting with large language models. In *ICRA*, 2025.

[93] Zijun Lin, Jiafei Duan, Haoquan Fang, Dieter Fox, Ranjay Krishna, Cheston Tan, and Bihan Wen. Failsafe: Reasoning and recovery from failures in vision-language-action models. *arXiv preprint arXiv:2510.01642*, 2025.

[94] Tingjun Dai, Mingfei Han, Tingwen Du, Zhiheng Liu, Zhihui Li, Salman Khan, Jun Yu, and Xiaojun Chang. See, plan, rewind: Progress-aware vision-language-action models for robust robotic manipulation. *arXiv preprint arXiv:2603.09292*, 2026.

[95] Changdae Oh, Seongheon Park, To Eun Kim, Jiatong Li, Wendi Li, Samuel Yeh, Xuefeng Du, Hamed Hassani, Paul Bogdan, Dawn Song, et al. Uncertainty quantification in llm agents: Foundations, emerging challenges, and opportunities. In *ACL*, 2026.

[96] Xingyao Lin, Xinghao Zhu, Tianyi Lu, Sicheng Xie, Hui Zhang, Xipeng Qiu, Zuxuan Wu, and Yu-Gang Jiang. Ask-to-clarify: Resolving instruction ambiguity through multi-turn dialogue. *arXiv preprint arXiv:2509.15061*, 2025.

[97] Wen-Han Hsieh, Elvis Hsieh, Dantong Niu, Trevor Darrell, Roei Herzig, and David M Chan. Do what? teaching vision-language-action models to reject the impossible. In *EMNLP Findings*, 2025.

Appendix

Contents

A Failure Score Visualization **16**

A.1 Failure Score Dynamics Across Splits 16

A.2 Qualitative Results 16

B Implementation Details **19**

B.1 Hyperparameters 19

B.2 Conformal Prediction 20

B.3 Evaluation Metrics 21

C Additional Ablation Studies **21**

C.1 Failure Onset Proxy 21

C.2 Thresholding Strategies 22

C.3 Layer-wise Ablation 23

C.4 Inference-Time Policy Steering 23

D Additional Analysis **24**

D.1 VLA Latent Visualization 24

D.2 Detection Timeliness Relative to Annotated Failure Onset 25

D.3 Detector Architecture 25

E Benchmark Details **26**

E.1 Simulation Setup 26

E.2 Real-Robot Setup 27

F GPT-5.2-based Failure Onset Annotation **28**

F.1 Annotation Protocol 28

F.2 Human Agreement Analysis 29

G Baselines **29**

G.1 Out-of-Distribution Detection-based Methods 30

G.2 Multi-Sampling-based Methods 30

G.3 Classifier-based Methods 31

G.4 Token Uncertainty-based Methods 32

G.5 VLM-based Method 32

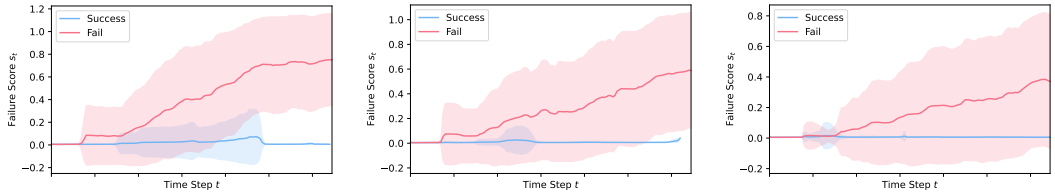
H Extended Related Work **33**

I Limitations and Future Work **33**

J Broader Impacts **33**

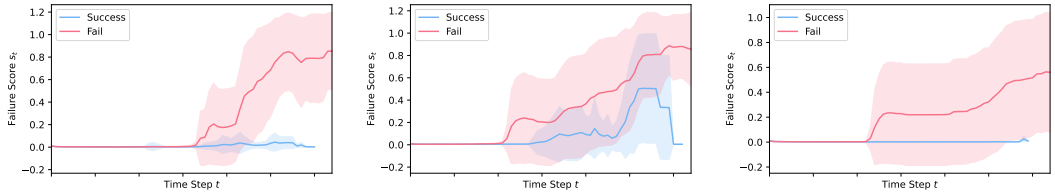
A Failure Score Visualization

A.1 Failure Score Dynamics Across Splits



(a) Train (b) Seen (c) Unseen

Figure 5: **Average failure score trend across episodes for OpenVLA on LIBERO-10.** The per-timestep score is averaged over successful (blue) and failure (red) trajectories on (a) the training set, (b) the seen evaluation split, and (c) the unseen evaluation split. Shaded regions indicate standard deviation across episodes.



(a) Train (b) Seen (c) Unseen

Figure 6: **Average failure score trend across episodes for π_0 on LIBERO-10.**

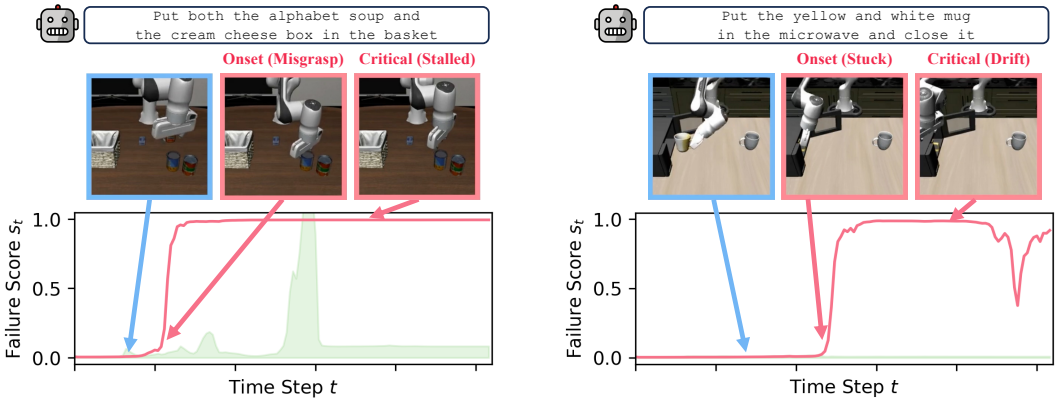
We visualize the average per-timestep failure score s_t across episodes, computed after training the detector, for the training set and the seen/unseen evaluation splits. Results are shown for OpenVLA (Figure 5) and π_0 (Figure 6) on LIBERO-10. Across all splits, the failure score remains consistently low for successful trajectories, while exhibiting a clear upward trend for failed trajectories. The consistent score dynamics across splits indicate that Hide-and-Seek learns transferable failure signals rather than overfitting to training-distribution cues.

Although Equation (2) propagates gradients only through the maximum-scoring timestep of each successful trajectory, the identity of this argmax varies across iterations and minibatch pairings. As a result, different timesteps receive supervision over the course of training. In practice, this effectively distributes supervision across most timesteps of successful trajectories: scores on successful rollouts remain consistently low across all splits, while those on failure rollouts stay well separated.

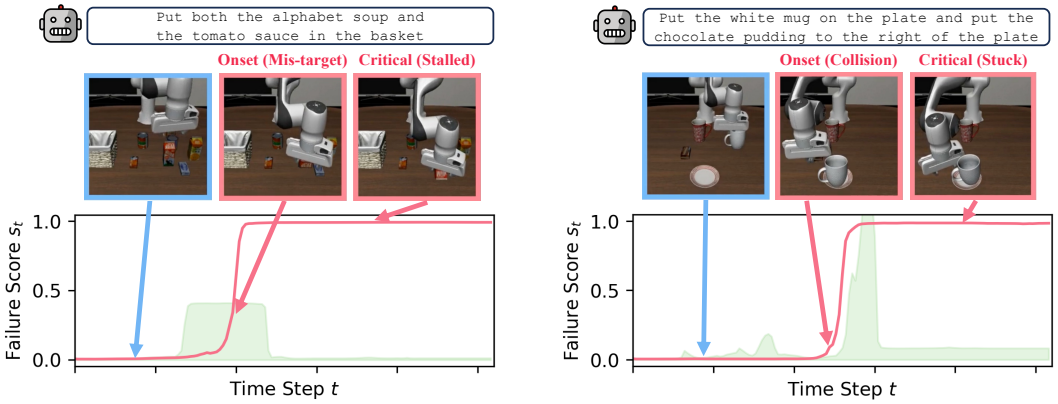
A.2 Qualitative Results

Hide-and-Seek's failure scores correlate with observable failure indicators in the visual scene, despite the detector being trained without any step-level supervision. The discovered onset point t_{onset} aligns with subtle early indicators of failure (*e.g.*, a book starting to slip from the gripper), while the peak point $t_{\max}$ aligns with the most salient and critical failure event (*e.g.*, the robot dropping the object). These moments correspond to meaningful transitions in the physical state of the task and match human intuition about when failure occurs, even though the model receives only trajectory-level outcome labels during training.

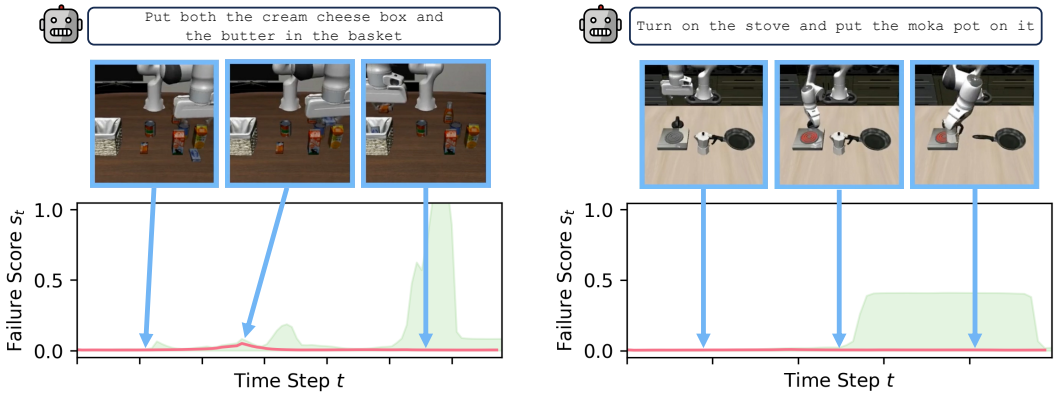
We visualize qualitative examples from OpenVLA (Figure 7) and π_0 (Figure 8) on LIBERO-10 tasks. A failure is declared at the earliest timestep t such that the failure score s_t (red curve) exceeds the threshold ζ_t (green region). The CP threshold ζ_t adapts to score variability across the calibration set, becoming higher when successful trajectories exhibit greater variation (*e.g.*, mid-execution) and lower when they are more consistent (*e.g.*, initialization). The failure score remains consistently low across successful trajectories and rises sharply at the onset of failure.



(a) **Failure Trajectories.** (left) Robot fails to grasp the alphabet soup and stops moving. (right) Robot gets stuck after placing the mug, then drifts away without closing the microwave.

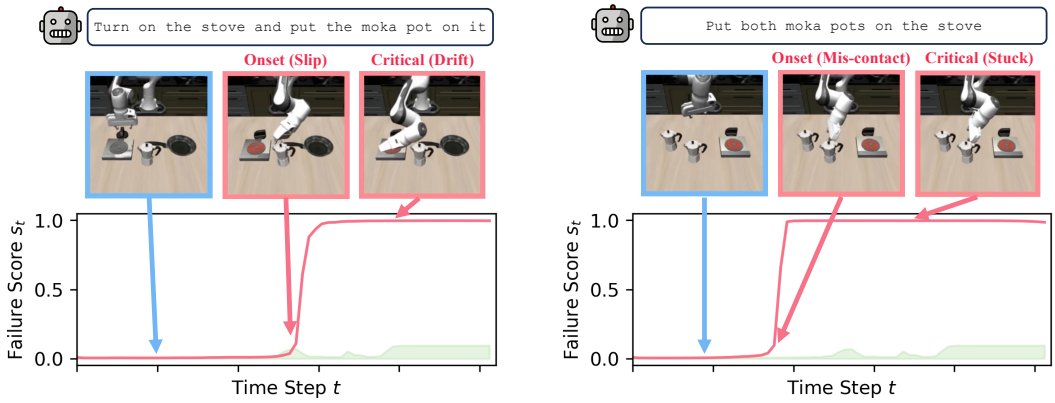


(b) **Failure Trajectories.** (left) Robot grasps the wrong object, drops it, and stalls. (right) Robot collides with the placed mug and gets stuck.

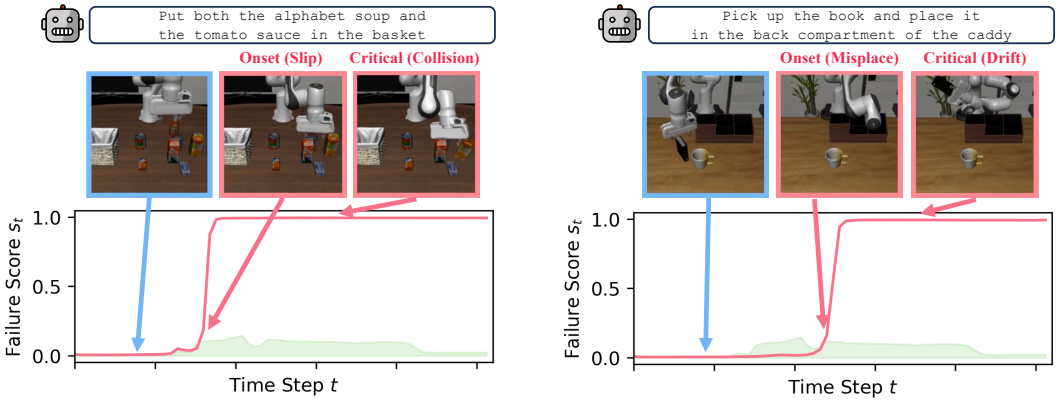


(c) **Successful Trajectories.** The failure score remains consistently low throughout the trajectory.

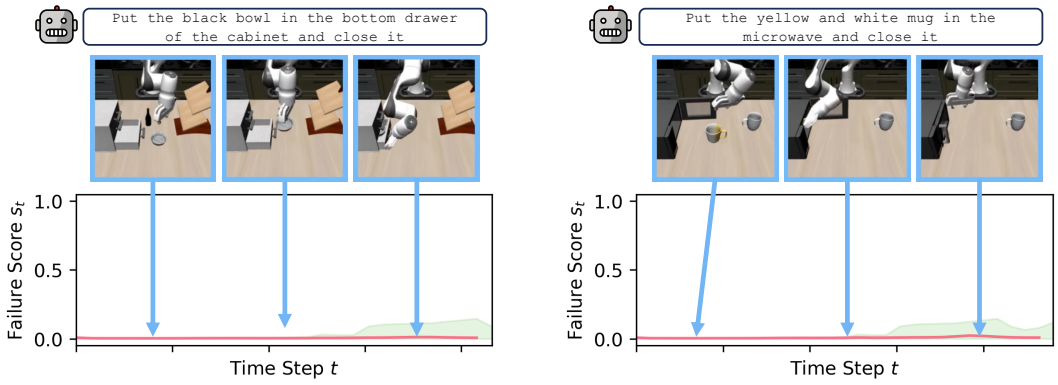
Figure 7: **Visualization of failure scores for the OpenVLA policy across different tasks.** A failure is declared at the earliest timestep t where the failure score s_t (red curve) exceeds the time-varying threshold ζ_t (green region) determined by conformal prediction.



(a) **Failure Trajectories.** (left) Robot slips the moka pot and drifts away. (right) Robot mis-contacts the top of the moka pot and gets stuck.



(b) **Failure Trajectories.** (left) Robot slips the alphabet soup and collides with nearby objects. (right) Robot misplaces the book on the caddy's divider and drifts away.



(c) **Successful Trajectories.** The failure score remains consistently low throughout the trajectory.

Figure 8: **Visualization of failure scores for the π_0 policy across different tasks.** A failure is declared at the earliest timestep t where the failure score s_t (red curve) exceeds the time-varying threshold ζ_t (green region) determined by conformal prediction.

B Implementation Details

We implement Hide-and-Seek in Python 3.10.19 with PyTorch 2.10.0 [66] on a single NVIDIA A6000 GPU with 48GB of memory, where training typically completes in approximately 5 minutes. We use the AdamW optimizer [67] with a learning rate of 5×10^{-4} and L2 parameter regularization with weight 0.1, applied uniformly across all simulation experiments. The failure detector is a single-layer LSTM with hidden size 256, followed by a linear head and a sigmoid output layer. We train for 300 epochs with a batch size of 64. To address class imbalance, we use a balanced sampler that ensures each mini-batch contains an equal number of failure and success trajectories.

We adopt a warmup schedule where $\mathcal{L}_{\text{inter}}$ is optimized alone for the first N_{warm} epochs until t_{onset} stabilizes as a reliable proxy for the failure onset, after which $\mathcal{L}_{\text{intra}}$ is introduced. We apply stop-gradient before computing t_{onset}, *i.e.*, $t_{\text{onset}} = \arg\max_t \text{sg}(s_t - s_{t-1})$ (implemented via `detach()`), so that gradients from $\mathcal{L}_{\text{intra}}$ flow only through the score values s_t in the pre/post-onset averages, not through the index selection.[4] Hyperparameter search space and selection details are provided in Section B.1. Importantly, we train a **single shared failure detector** jointly across multiple tasks and evaluate the same detector on the corresponding multi-task test set.

B.1 Hyperparameters

We adopt a two-stage hyperparameter tuning procedure for the simulation benchmarks. We first perform a grid search over global hyperparameters shared across experiments—including optimizer settings and model architecture—on LIBERO-10 with OpenVLA, summarized in Table 7. Given this configuration, we then tune benchmark- and policy-specific hyperparameters for each VLA model independently, as reported in Table 8. For real-robot experiments, we use a shared set of hyperparameters across both task categories, listed in Table 9. All tuning is performed exclusively on seen tasks; the resulting models are deployed zero-shot on unseen tasks without further adaptation, and we report held-out evaluation results on both splits. Baselines are tuned under the same protocol for fair comparison.

Table 7: Hyperparameter search space applied uniformly across all experiments. The selected value is **bolded**.

Hyperparameter	**Search Space**
Optimizer	
Optimizer	{Adam, **AdamW**, SGD}
Learning rate	{1e-2, 5e-3, 1e-3, **5e-4**, 1e-4}
L2 regularization	{1, **1e-1**, 1e-2, 1e-3, 1e-4}
Training epoch	{100, **300**, 500, 1000}
Warmup epoch	{0, **50**, 100, 150}
Architecture	
LSTM hidden size	{128, **256**, 512}
Context length	{1, 4, 8, **16**, all}

Table 8: Hyperparameter search space and selected values per policy on simulation benchmarks.

Hyperparameter	**Search Space**	**OpenVLA (LIBERO)**	**π_0 (LIBERO)**	**$\pi_{0.5}$ (VLABench)**
Temporal Aggregation				
Window size w	{1, 4, 5, 8, 16}	4	8†	5†
Loss				
Inter loss margin m_r	{0.1, 0.2, 0.3, 0.5, 1.0}	1.0	1.0	1.0
Intra loss margin m_o	{0.1, 0.2, 0.3, 0.5, 1.0}	0.2	0.5	0.2
Intra weight λ	{1e-3, 5e-3, 1e-2, 1e-1, 1}	1e-1	5e-3	1e-1

† Set to match the policy's action chunk size (replanning horizon).

[4] We constrain $t_{\text{onset}} \in [2, N_{\tau_f}]$ to ensure both summations in Equation (3) are well-defined, although this boundary case barely occurs in practice.

Table 9: Hyperparameter search space for real-robot experiments. The selected values are **bolded**.

Hyperparameter	Search Space
Optimizer	
Optimizer	{Adam, **AdamW**, SGD}
Learning rate	{1e-2,5e-3,1e-3,**5e-4**,1e-4}
L2 regularization	{1, 1e-1, **1e-2**, 1e-3, 1e-4}
Training epochs	{100, **300**, 500, 1000}
Warmup epochs	{0, **50**, 100, 150}
Architecture	
LSTM hidden size	{128, **256**, 512}
Context length	{1, 4, **8**, 16, all}
Temporal Aggregation	
Window size w	{1, 4, **8**, 16}
Loss	
Inter loss margin m_r	{0.1, 0.2, 0.3, 0.5, **1.0**}
Intra loss margin m_o	{0.1, **0.2**, 0.3, 0.5, 1.0}
Intra weight λ	{1e-3, **5e-3**, 1e-2, 1e-1, 1}

B.2 Conformal Prediction

To convert per-step failure scores s_t into binary alarms, we calibrate a time-varying threshold ζ_t using functional conformal prediction [53]. Calibration is performed on a held-out set of C successful trajectories $\mathcal{D}_{\text{cal}} = \{\tau^{(c)}\}_{c=1}^{C}$, padded to a common length T by repeating the final score. Under exchangeability and a user-specified significance level $\alpha \in (0, 1)$, the resulting band guarantees that the trajectory-level false-alarm rate on new successful rollouts is bounded by α. We partition $\mathcal{D}_{\text{cal}}$ into two disjoint subsets $\mathcal{D}_{\text{cal}_A}$ and $\mathcal{D}_{\text{cal}_B}$ of sizes N_1 and N_2, respectively[5], where the first estimates the score profile of successful execution and the second calibrates the band width. On $\mathcal{D}_{\text{cal}_A}$, we compute the per-step mean failure score:

$$\mu_t = \frac{1}{N_1} \sum_{i=1}^{N_1} s_t^{(i)}, \quad t = 1, \ldots, T, \tag{6}$$

and define a modulation function $\sigma(t)$ that captures the typical spread of successful trajectories around μ_t. We adopt the adaptive variant from [53]:

$$\sigma(t) = \max_{k \in \mathcal{H}} \left| s_t^{(k)} - \mu_t \right|, \tag{7}$$

where $\mathcal{H} \subseteq [N_1]$ is the set of trajectories whose maximum residual falls within the $(1-\alpha)$-quantile of $\{\max_{t \in [T]} |s_t^{(m)} - \mu_t|\}_{m=1}^{N_1}$, excluding outliers; when $(N_1 + 1)(1 - \alpha) > N_1$, we set $\mathcal{H} = [N_1]$. For each calibration trajectory $j \in \mathcal{D}_{\text{cal}_B}$, we compute the maximum modulated deviation:

$$D_j = \max_{t \in [T]} \frac{s_t^{(j)} - \mu_t}{\sigma(t)}, \tag{8}$$

using a one-sided formulation since failures manifest as upward deviations. Letting h denote the $(1-\alpha)$-quantile of $\{D_j\}_{j=1}^{N_2}$, the time-varying threshold is:

$$\zeta_t = \mu_t + h \cdot \sigma(t), \tag{9}$$

which corresponds to $\zeta_t = \mu_t + b_t$ in Section 4.2 with bandwidth $b_t = h \cdot \sigma(t)$. At runtime, a failure is declared at the earliest timestep t for which $s_t \geq \zeta_t$. As no single α is universally optimal, we report results averaged over $\alpha \in \{0.15, 0.20, 0.25\}$ in our main experiments. For the accuracy–timeliness analysis in Figure 3, we sweep $\alpha \in [0.01, 1.0]$ with a step size of 0.01, plotting the corresponding average normalized detection time and balanced accuracy for each operating point, following [15].

[5] In our implementation, N_1 and N_2 are 30% and 70% of the calibration set, respectively.

For visualization clarity, we truncate the x-axis at the maximum detection-time endpoint reached by our method.

We note that the false-alarm guarantee of functional conformal prediction holds under the assumption of exchangeability between calibration and test trajectories. In the unseen-task setting, where calibration is performed using successful trajectories from seen tasks, this assumption holds approximately. Since unseen tasks are unavailable at calibration time, this setup is the standard practice in prior runtime monitoring work [12, 15]. Empirically, we observe that failure score profiles on successful trajectories remain stable and consistent across seen and unseen tasks (see Figures 5 and 6), suggesting that the calibrated thresholds transfer well in practice. Consistent with this, our method maintains strong performance on unseen tasks (Tables 1 to 3).

B.3 Evaluation Metrics

We provide full definitions of the evaluation metrics for runtime failure detection. We treat failure trajectories as positives and successful trajectories as negatives; a true positive is recorded if the detector raises an alarm at any timestep within a failure trajectory. An alarm is triggered when the detector score exceeds a time-varying threshold ζ_t obtained via functional conformal prediction.

Balanced Accuracy (bACC). Balanced accuracy equally weights positive and negative classes:

$$\text{bACC} = \frac{1}{2}(\text{TPR} + \text{TNR}), \tag{10}$$

where TPR (true positive rate) is the fraction of failure trajectories correctly detected, and TNR (true negative rate) is the fraction of success trajectories with no false alarm. A true positive is counted if an alarm is raised at any timestep within a failure trajectory.

Weighted Accuracy (wACC). To account for class imbalance, weighted accuracy assigns weights based on the empirical class ratio:

$$\beta = \frac{\#\text{Successful Rollouts}}{\#\text{Rollouts}}, \quad \text{wACC} = \beta \cdot \text{TNR} + (1-\beta) \cdot \text{TPR}. \tag{11}$$

This formulation reflects deployment scenarios where class proportions may be imbalanced.

Time-weighted Accuracy (TWA). Time-weighted accuracy [16] jointly captures correctness and timeliness of detection. Let t_i (i.e., detection time) denote the first alarm timestep for the i-th true positive trajectory, and let T_i denote its trajectory length. Then:

$$\text{TWA} = \frac{1}{2}\left(\frac{1}{|\mathcal{P}|}\sum_{i \in \mathcal{TP}}\left(1 - \frac{t_i}{T_i}\right) + \frac{|\mathcal{TN}|}{|\mathcal{N}|}\right). \tag{12}$$

where $\mathcal{TP}$ and $\mathcal{TN}$ denote the sets of true positives and true negatives, respectively, and $\mathcal{P}, \mathcal{N}$ are the sets of positive and negative trajectories. Earlier detections yield higher scores, while delayed alarms are penalized proportionally to their detection time.

C Additional Ablation Studies

C.1 Failure Onset Proxy

A key empirical finding of our method is that training with $\mathcal{L}_{\text{inter}}$ implicitly discovers the failure onset through t_{onset} without any temporal supervision. To verify this, we compare three characteristic timesteps: the peak score timestep $t_{\max} = \arg\max_t s_t$, which is directly optimized by $\mathcal{L}_{\text{inter}}$; the spike point $t_{\text{onset}} = \arg\max_t(s_t - s_{t-1})$, which receives no direct gradient; and the GPT-5.2 annotated failure onset that serves as the oracle reference. All timesteps are normalized by trajectory length.

As shown in Table 10, $t_{\max}$ lags behind the oracle onset ($+0.107$ for OpenVLA, $+0.068$ for π_0), as it reflects the timestep of the most failure-indicative signal rather than the failure onset itself. Anchoring $\mathcal{L}_{\text{intra}}$ at $t_{\max}$ therefore degrades both detection accuracy and timeliness: the loss mistakenly treats

Table 10: Effect of different proxy failure onset definitions in the intra-trajectory contrastive loss $\mathcal{L}_{\text{intra}}$ on failure detection performance on LIBERO-10. Estimated Onset denotes the average normalized timestep of the selected proxy.

Policy: OpenVLA (Success Rate: 51.0%)							
		Seen Tasks			**Unseen Tasks**		
Proxy	**Estimated Onset**	bACC↑	wACC↑	TWA↑	bACC↑	wACC↑	TWA↑
Oracle	0.401	$0.854^{\pm 0.027}$	$0.872^{\pm 0.033}$	$0.673^{\pm 0.028}$	$0.836^{\pm 0.041}$	$0.835^{\pm 0.068}$	$0.668^{\pm 0.012}$
$t_{\max}$	0.508	$0.845^{\pm 0.022}$	$0.839^{\pm 0.021}$	$0.612^{\pm 0.020}$	$0.803^{\pm 0.046}$	$0.796^{\pm 0.040}$	$0.591^{\pm 0.044}$
t_{onset}	0.417	$0.852^{\pm 0.051}$	$0.853^{\pm 0.052}$	$0.660^{\pm 0.035}$	$0.834^{\pm 0.036}$	$0.828^{\pm 0.034}$	$0.663^{\pm 0.010}$
Policy: π_0 (Success Rate: 84.2%)							
		Seen Tasks			**Unseen Tasks**		
Proxy	**Estimated Onset**	bACC↑	wACC↑	TWA↑	bACC↑	wACC↑	TWA↑
Oracle	0.478	$0.853^{\pm 0.097}$	$0.902^{\pm 0.093}$	$0.686^{\pm 0.063}$	$0.924^{\pm 0.022}$	$0.981^{\pm 0.016}$	$0.684^{\pm 0.046}$
$t_{\max}$	0.546	$0.859^{\pm 0.056}$	$0.899^{\pm 0.076}$	$0.613^{\pm 0.030}$	$0.854^{\pm 0.092}$	$0.848^{\pm 0.093}$	$0.627^{\pm 0.062}$
t_{onset}	0.470	$0.885^{\pm 0.041}$	$0.926^{\pm 0.054}$	$0.693^{\pm 0.021}$	$0.892^{\pm 0.011}$	$0.921^{\pm 0.044}$	$0.705^{\pm 0.043}$

actions between the true onset and $t_{\max}$ as pre-onset references, suppressing the score rise that should occur at the actual onset and pushing detection later in the trajectory.

In contrast, t_{onset} closely approximates the oracle for both OpenVLA ($+0.016$) and π_0 (-0.008) despite receiving no direct gradient, and using it as the anchor achieves performance on par with the oracle itself and substantially better than $t_{\max}$. This confirms that the sharpest score transition aligns with the true failure onset, motivating our intra-trajectory onset contrast loss $\mathcal{L}_{\text{intra}}$ to directly optimize the score separation around t_{onset}.

C.2 Thresholding Strategies

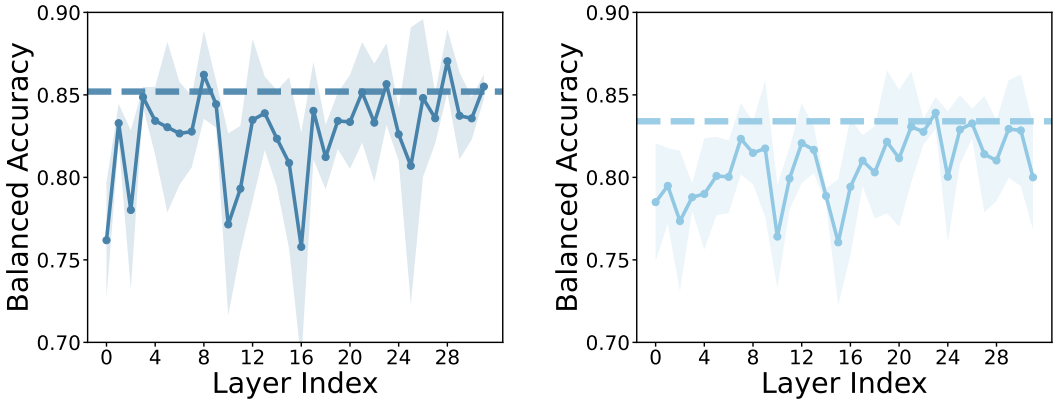
Table 11: Effect of different thresholding strategies on failure detection performance on LIBERO-10. We report the mean and standard deviation over three random seeds.

Policy: OpenVLA (Success Rate: 51.0%)						
	Seen Tasks			**Unseen Tasks**		
Thresholding	bACC↑	wACC↑	TWA↑	bACC↑	wACC↑	TWA↑
Fixed ($\zeta = 0.5$)	$0.859^{\pm 0.035}$	$0.864^{\pm 0.034}$	$0.656^{\pm 0.024}$	$0.792^{\pm 0.032}$	$0.780^{\pm 0.026}$	$0.625^{\pm 0.029}$
Split CP	$0.850^{\pm 0.030}$	$0.842^{\pm 0.028}$	$0.656^{\pm 0.011}$	$0.833^{\pm 0.026}$	$0.839^{\pm 0.022}$	$0.658^{\pm 0.005}$
Functional CP	$0.852^{\pm 0.051}$	$0.853^{\pm 0.052}$	$0.660^{\pm 0.035}$	$0.834^{\pm 0.036}$	$0.828^{\pm 0.034}$	$0.663^{\pm 0.010}$
Policy: π_0 (Success Rate: 84.2%)						
	Seen Tasks			**Unseen Tasks**		
Thresholding	bACC↑	wACC↑	TWA↑	bACC↑	wACC↑	TWA↑
Fixed ($\zeta = 0.5$)	$0.827^{\pm 0.101}$	$0.730^{\pm 0.173}$	$0.611^{\pm 0.056}$	$0.707^{\pm 0.092}$	$0.716^{\pm 0.108}$	$0.609^{\pm 0.032}$
Split CP	$0.729^{\pm 0.083}$	$0.859^{\pm 0.070}$	$0.605^{\pm 0.092}$	$0.703^{\pm 0.097}$	$0.890^{\pm 0.107}$	$0.620^{\pm 0.093}$
Functional CP	$0.885^{\pm 0.041}$	$0.926^{\pm 0.054}$	$0.693^{\pm 0.021}$	$0.892^{\pm 0.011}$	$0.921^{\pm 0.044}$	$0.705^{\pm 0.043}$

We compare three thresholding strategies: (1) a fixed threshold of 0.5 requiring no calibration set, (2) a time-invariant scalar threshold calibrated via split conformal prediction (Split CP) [68], and (3) a time-varying threshold from functional conformal prediction (Functional CP), which we adopt as our primary strategy. We report the mean across $\alpha \in \{0.15, 0.20, 0.25\}$.

We observe that our method is relatively robust to the choice of thresholding strategy under the OpenVLA policy, with only minor performance variations across methods. In contrast, both fixed and Split CP thresholds degrade noticeably under the π_0 policy, indicating higher sensitivity to threshold selection in this setting. Functional CP consistently delivers more stable performance across both policies, suggesting improved robustness to policy-specific score distributions and better alignment with the sequential nature of the decision process.

C.3 Layer-wise Ablation



(a) Seen tasks

(b) Unseen tasks

Figure 9: Layer-wise ablation of OpenVLA internal representations for failure detection on LIBERO-10. We report balanced accuracy (bACC) across layer indices. The dashed line corresponds to using layer-averaged representations.

We ablate which transformer layer of OpenVLA provides the most informative representation for failure detection by training Hide-and-Seek on each layer's embedding $h_t^{(\ell)} \in \mathbb{R}^d$ for $\ell \in \{0, 1, \ldots, 31\}$, and compare against our default of averaging across all layers.

As shown in Figure 9, performance varies across layers, but does not exhibit a sharp peak at any single layer. Mid-to-late layers (e.g., layers 20–31) generally yield stronger and more stable performance than early layers, consistent with the intuition that deeper representations capture more task-relevant abstractions. However, the performance gap among well-performing layers is small (within $\sim 3\%$ bACC), and the layer-averaged representation (dashed line) matches or exceeds the best individual layer on both seen (85.2%) and unseen (83.4%) tasks. We therefore adopt layer averaging as our default since it eliminates the need for per-policy layer selection while providing consistent performance across both task splits.

It is worth noting that recent work in LLM/LVLM interpretability has reported that intermediate layers often encode the most semantically rich features for downstream tasks [69, 70]. While our results show a similar mild advantage for mid-to-late layers, a more principled investigation of which layers encode failure-discriminative information across different VLA architectures and policy types is an interesting direction for future work.

C.4 Inference-Time Policy Steering

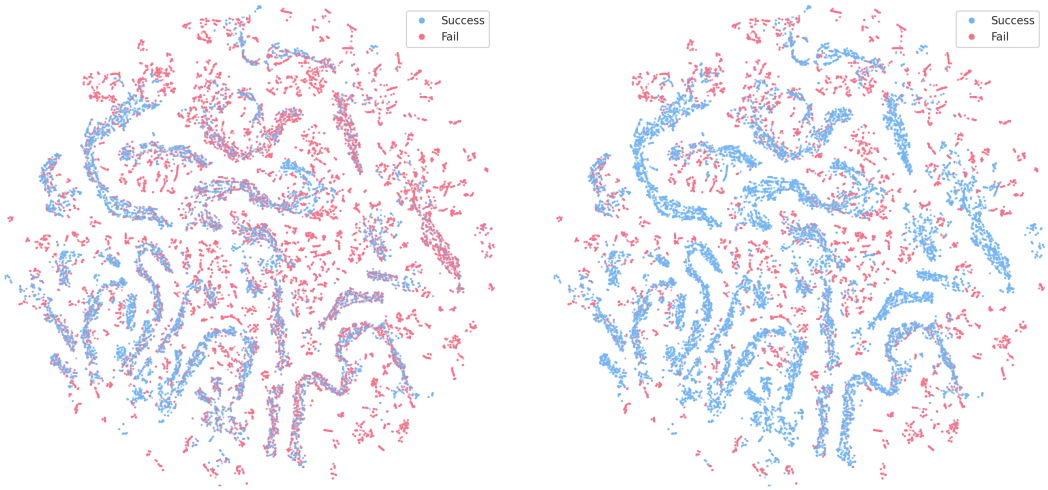
Beyond runtime failure monitoring, we investigate whether Hide-and-Seek can also serve as an inference-time verifier for closed-loop policy improvement [71, 72]. Specifically, on the held-out LIBERO-10 test split, we sample N candidate action chunks from the frozen π_0 policy at each decision step, score each candidate using the failure score predicted by Hide-and-Seek, and execute the candidate with the lowest predicted failure likelihood. This procedure requires neither policy fine-tuning nor additional robot data. As shown in Table 12, verifier-guided action selection improves the task success rate from 83.4% to 87.4%, corresponding to a $+4.0$ percentage-point improvement over the base policy. These results suggest that the failure signals discovered by Hide-and-Seek are useful not only for detecting impending failures, but also for distinguishing among candidate actions during execution. Thus, Hide-and-Seek can naturally support inference-time policy steering and may provide a useful signal for more general closed-loop mechanisms such as recovery, replanning, or human intervention.

Table 12: Success rate (%) with verifier-guided action selection on the held-out LIBERO-10 test split.

Method	**SR (%)**
π_0	83.4
π_0 + Hide-and-Seek	**87.4**

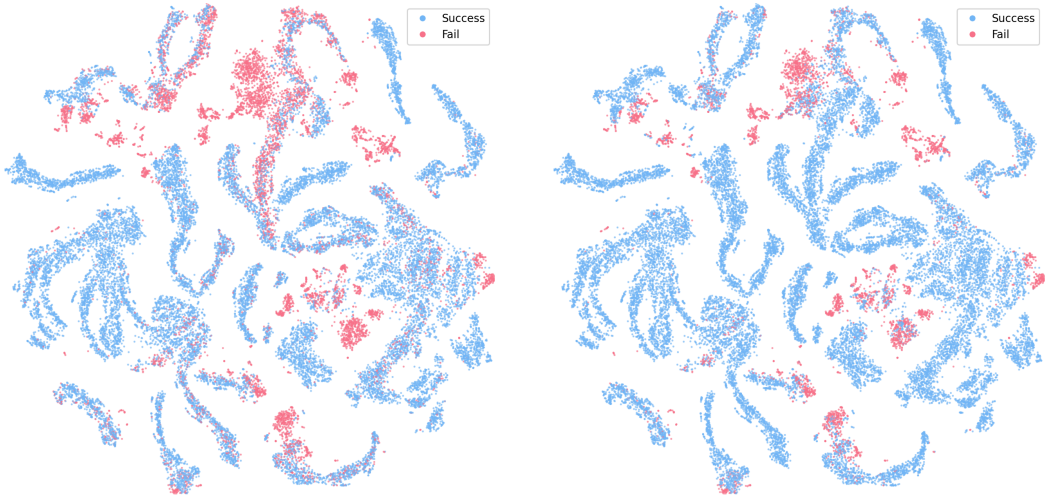
D Additional Analysis

D.1 VLA Latent Visualization



(a) Uniform trajectory-level labeling (b) Onset-aware relabeling

Figure 10: **Visualization of OpenVLA action embeddings on LIBERO-10.** (a) Under uniform trajectory-level labeling, all timesteps in failure trajectories are assigned the failure label, mislabeling a substantial portion of normal actions as failures, resulting in significant overlap between success (blue) and failure (red) embeddings. (b) Relabeling pre-onset timesteps as success yields a clearer separation between the two classes.



(a) Uniform trajectory-level labeling (b) Onset-aware relabeling

Figure 11: **Visualization of π_0 action embeddings on LIBERO-10.**

To better understand the impact of supervision granularity, we visualize the internal VLA action embeddings described in Section 5.1. We collect rollouts from all tasks in LIBERO-10 using OpenVLA and π_0, resulting in 500 episodes per policy (50 per task). At each timestep, we extract a compact representation $h_t \in \mathbb{R}^d$: for OpenVLA, we average the action embedding across layer and DoF dimensions; for π_0, we use the hidden state before the velocity prediction head at the final denoising step. We then project the embeddings to 2D using t-SNE [73] for visualization.

Figures 10a and 11a visualize the embeddings under the standard trajectory-level supervision, where all timesteps in a failure trajectory are assigned the same failure label. We observe substantial overlap between success and failure embeddings, indicating that many normal actions are mislabeled as failure. This suggests that uniformly propagating trajectory-level labels introduces significant noise and blurs the decision boundary. To isolate failure-relevant signals, we refine the labels within failure trajectories using the failure onset annotations described in Section F, relabeling timesteps preceding the onset as success. As shown in Figures 10b and 11b, this results in a much clearer separation between success and failure embeddings.

These observations provide empirical support for our formulation: uniformly propagating trajectory-level labels dilutes discriminative signals by assigning failure labels to predominantly correct actions, whereas isolating the failure phase yields a more separable representation. This motivates our objective, which automatically discovers failure-indicative actions without requiring step-level annotations.

D.2 Detection Timeliness Relative to Annotated Failure Onset

Table 13: **Detection lag and balanced accuracy on LIBERO-10.** Detection lag $\Delta t = \frac{1}{|\mathcal{TP}|}\sum_i (t_i^* - t_{\text{det},i})/T_i$ is computed over true-positive failure trajectories; $\Delta t > 0$ indicates early alarms and $\Delta t < 0$ indicates delayed alarms. Results are averaged over $\alpha \in \{0.15, 0.20, 0.25\}$ and three random seeds.

	OpenVLA				π_0			
	Seen Tasks		Unseen Tasks		Seen Tasks		Unseen Tasks	
Method	**Lag Δt**	**bACC↑**	**Lag Δt**	**bACC↑**	**Lag Δt**	**bACC↑**	**Lag Δt**	**bACC↑**
SAFE-LSTM	$+0.035^{\pm 0.054}$	$0.670^{\pm 0.100}$	$-0.147^{\pm 0.087}$	$0.686^{\pm 0.133}$	$+0.196^{\pm 0.091}$	$0.774^{\pm 0.024}$	$+0.037^{\pm 0.102}$	$0.609^{\pm 0.057}$
SAFE-MLP	$-0.057^{\pm 0.045}$	$0.823^{\pm 0.036}$	$-0.272^{\pm 0.189}$	$0.775^{\pm 0.062}$	$+0.121^{\pm 0.102}$	$0.870^{\pm 0.038}$	$-0.052^{\pm 0.154}$	$0.801^{\pm 0.083}$
Ours	$+0.042^{\pm 0.043}$	$0.852^{\pm 0.051}$	$+0.019^{\pm 0.016}$	$0.834^{\pm 0.036}$	$+0.078^{\pm 0.030}$	$0.885^{\pm 0.041}$	$+0.052^{\pm 0.093}$	$0.892^{\pm 0.011}$

A reliable failure detector should raise alarms close to the actual moment of failure (neither too early nor too late), enabling timely intervention without overly aggressive interruptions. We quantify this via the normalized detection lag relative to the GPT-5.2 annotated failure onset t^* (Section F):

$$\Delta t = \frac{1}{|\mathcal{TP}|} \sum_{i \in \mathcal{TP}} \frac{t_i^* - t_{\text{det},i}}{T_i},$$

where the detection time $t_{\text{det},i}$ is the first timestep at which $s_t \geq \zeta_t$, and T_i is the trajectory length. A positive $\Delta t > 0$ indicates *early* alarms, while a negative $\Delta t < 0$ indicates *delayed* alarms.

As shown in Table 13, we compare against the strongest classifier-based baseline SAFE. **Hide-and-Seek consistently raises alarms slightly before the annotated failure onset across all settings, despite being trained without step-level supervision.** This enables timely intervention—such as pausing execution or requesting human assistance—before the policy becomes irrecoverably stuck or causes harm in real-world deployment.

In contrast, SAFE-MLP exhibits negative lag in three of four settings (e.g., -0.272 on OpenVLA unseen), indicating delayed detection after failure onset. We attribute this to its cumulative score formulation with a linearly increasing target weight (Eq. 23), which biases scores toward later timesteps regardless of when failure occurs. SAFE-LSTM produces more timely alarms on average, but with higher variance and lower accuracy. Overall, Hide-and-Seek achieves both the highest bACC and consistently timely detection across all settings.

D.3 Detector Architecture

We provide detailed results and hyperparameter settings for the backbone architecture ablation (summarized in Table 4) in Table 14. We compare our LSTM-based design against three common alternatives: an MLP, a GRU [63], and a single-layer Transformer [64]. All variants share the same input features, training objective, and schedule, differing only in the backbone architecture. The MLP performs noticeably worse, highlighting the importance of temporal context for detecting failures in VLA policies.

In contrast, all context-aware architectures (GRU, Transformer, LSTM) achieve substantially stronger performance, indicating that our method generalizes across sequence models rather than relying on

Table 14: Architecture ablation of the failure detector on LIBERO-10. All variants share the same training objective and differ only in backbone architecture. LR denotes the learning rate, and Reg denotes the L2 regularization weight.

Policy: OpenVLA (Success Rate: 51.0%)								
	Hyperparameters		**Seen Tasks**			**Unseen Tasks**		
Arch	**LR**	**Reg**	**bACC↑**	**wACC↑**	**TWA↑**	**bACC↑**	**wACC↑**	**TWA↑**
MLP	1e-4	1e-1	$0.757^{\pm 0.027}$	$0.757^{\pm 0.028}$	$0.578^{\pm 0.027}$	$0.772^{\pm 0.064}$	$0.742^{\pm 0.084}$	$0.588^{\pm 0.020}$
GRU	5e-4	1e-1	$0.844^{\pm 0.043}$	$0.830^{\pm 0.049}$	$0.639^{\pm 0.031}$	$0.822^{\pm 0.068}$	$0.796^{\pm 0.072}$	$0.623^{\pm 0.028}$
Transformer	5e-4	1e-4	$0.818^{\pm 0.021}$	$0.818^{\pm 0.022}$	$0.627^{\pm 0.021}$	$0.788^{\pm 0.077}$	$0.753^{\pm 0.104}$	$0.621^{\pm 0.030}$
LSTM	5e-4	1e-1	$\mathbf{0.852}^{\pm 0.051}$	$\mathbf{0.853}^{\pm 0.052}$	$\mathbf{0.660}^{\pm 0.035}$	$\mathbf{0.834}^{\pm 0.036}$	$\mathbf{0.828}^{\pm 0.034}$	$\mathbf{0.663}^{\pm 0.010}$
Policy: π_0 (Success Rate: 84.2%)								
	Hyperparameters		**Seen Tasks**			**Unseen Tasks**		
Arch	**LR**	**Reg**	**bACC↑**	**wACC↑**	**TWA↑**	**bACC↑**	**wACC↑**	**TWA↑**
MLP	5e-4	1e-3	$0.770^{\pm 0.087}$	$0.842^{\pm 0.093}$	$0.576^{\pm 0.079}$	$0.714^{\pm 0.041}$	$0.729^{\pm 0.201}$	$0.563^{\pm 0.017}$
GRU	1e-4	1e-1	$0.868^{\pm 0.090}$	$0.892^{\pm 0.061}$	$0.686^{\pm 0.059}$	$\mathbf{0.916}^{\pm 0.024}$	$\mathbf{0.972}^{\pm 0.027}$	$0.696^{\pm 0.033}$
Transformer	1e-4	1e-4	$0.836^{\pm 0.067}$	$0.914^{\pm 0.043}$	$0.638^{\pm 0.085}$	$0.811^{\pm 0.034}$	$0.853^{\pm 0.120}$	$0.646^{\pm 0.016}$
LSTM	5e-4	1e-1	$\mathbf{0.885}^{\pm 0.041}$	$\mathbf{0.926}^{\pm 0.054}$	$\mathbf{0.693}^{\pm 0.021}$	$0.892^{\pm 0.011}$	$0.921^{\pm 0.044}$	$\mathbf{0.705}^{\pm 0.043}$

a specific backbone. Among them, the LSTM performs best, achieving TWA gains of 2.1 and 3.3 points over GRU and Transformer on seen tasks, and 4.0 and 4.2 points on unseen tasks, respectively, on OpenVLA. Overall, these results suggest that while our approach is broadly compatible with context-aware architectures, the LSTM provides the best performance in our setting.

E Benchmark Details

Training and evaluation protocol. For both simulation (Section E.1) and real-robot (Section E.2) experiments, we train the failure detector on the training set and evaluate on a held-out evaluation set. For conformal prediction, we calibrate thresholds using held-out successful trajectories: (1) for seen-task evaluation, we use a calibration set from seen tasks; (2) for unseen-task evaluation, we use successful trajectories from the seen-task evaluation split as the calibration set. All tuning is performed on seen tasks; **models are deployed zero-shot on unseen tasks** without further adaptation, and we report held-out results on both splits. Results are averaged over three random seeds, each with a different random partition of tasks into seen and unseen subsets. Dataset statistics are provided in Table 15.

Table 15: Benchmark statistics for each split.

	Number of Tasks			**Number of Rollouts**			
Benchmark	**Seen**	**Unseen**	**Total**	**Train**	**Calib Seen**	**Eval Seen**	**Eval Unseen**
LIBERO	7	3	10	210	35	105	150
VLABench	7	3	10	210	35	105	150
Real Cube	3	1	4	72	12	36	40
Real Kitchen	3	1	4	72	12	36	40

E.1 Simulation Setup

We evaluate on two simulation benchmarks. **LIBERO** [25] is a widely adopted benchmark for VLA evaluation; we use the LIBERO-10 suite, which consists of long-horizon tasks with diverse objects, layouts, and language instructions. We evaluate OpenVLA [1] and π_0 [2] using fine-tuned checkpoints released by the original authors. **VLABench** [26] is a recent benchmark covering diverse manipulation tasks; we evaluate $\pi_{0.5}$ [27] fine-tuned on its primary tasks. For both simulation benchmarks, we randomly hold out 3 out of 10 tasks as unseen for evaluation.

E.2 Real-Robot Setup

Real-robot experiments are conducted on a **UFactory xArm 6** with 8 tasks grouped into two categories (**CUBE** and **KITCHEN**) of 4 tasks each, listed in Table 16, with sequential execution snapshots visualized in Table 17. We collect 25 demonstrations per task to fine-tune $\pi_{0.5}$ [27] using LoRA [74]. For each category, we collect diverse rollouts by randomizing robot initial states and object configurations, and *all failure trajectories arise naturally from the policy's own execution*—we apply no human-induced perturbations or adversarial interventions, ensuring the failure distribution reflects the policy's genuine deployment behavior. This yields 98 successful and 62 failure trajectories for CUBE, and 102 successful and 58 failure trajectories for KITCHEN. Within each category, we designate 3 tasks as seen and 1 task as unseen for evaluating generalization.

Table 16: List of tasks used in the real-world experiments.

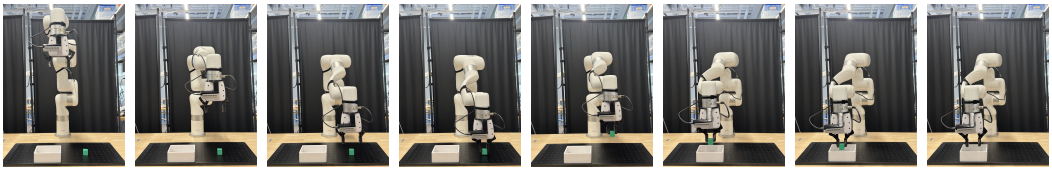
Task	Category	Instruction
1	Cube	Place the yellow cube into the container.
2		Place the green cube into the container.
3		Place both the blue cube and the green cube into the container.
4		Stack the red cube on top of the blue cube.
5	Kitchen	Place the duck doll in the bowl.
6		Place the carrot in the bowl.
7		Place both the carrot and the duck in the bowl.
8		Place the carrot in the bowl, then place the bowl on the plate.

Table 17: Sequential execution snapshots for real-robot tasks. Each task is shown as 8 ordered frames (left-to-right). Tasks are grouped into two categories: CUBE (Tasks 1–4) and KITCHEN (Tasks 5–8).

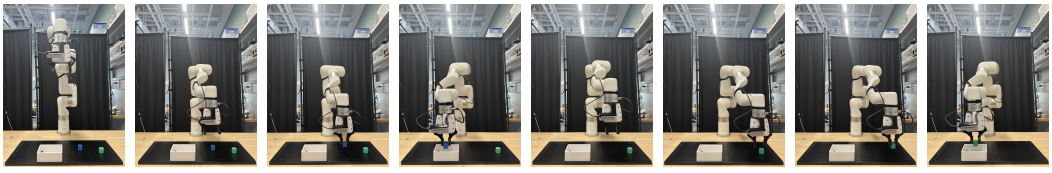
CATEGORY 1: CUBE



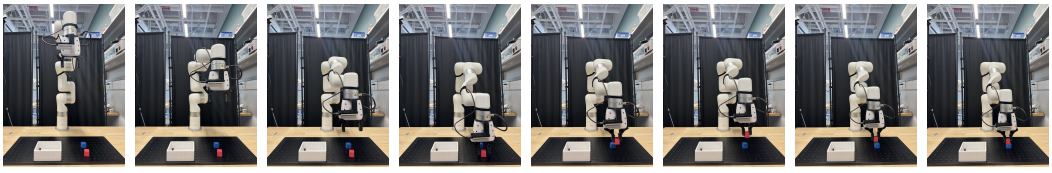
Task 1. Place the yellow cube into the container.



Task 2. Place the green cube into the container.

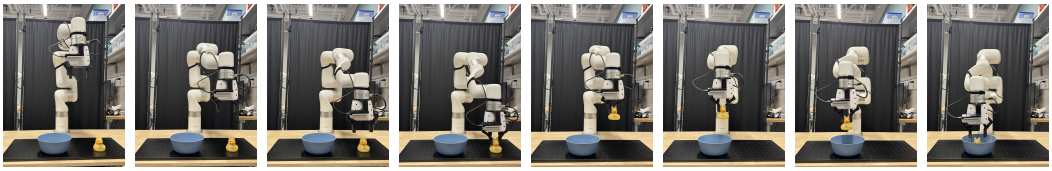


Task 3. Place both the blue cube and the green cube into the container.

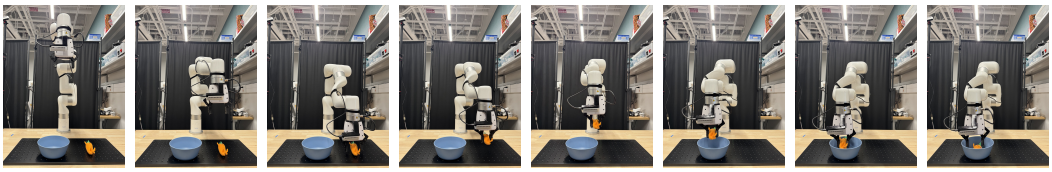


Task 4. Stack the red cube on top of the blue cube.

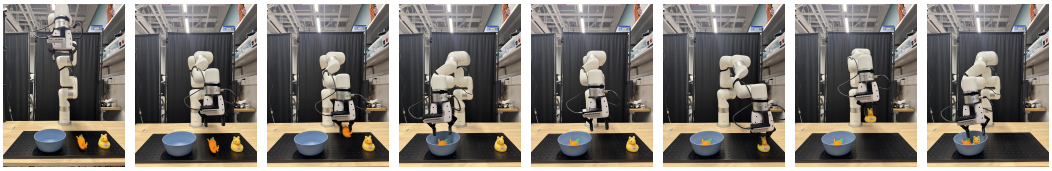
CATEGORY 2: KITCHEN



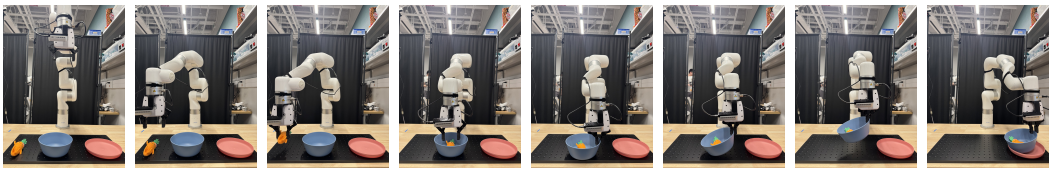
Task 5. Place the duck doll in the bowl.



Task 6. Place the carrot in the bowl.



Task 7. Place both the carrot and the duck in the bowl.



Task 8. Place the carrot in the bowl, then place the bowl on the plate.

F GPT-5.2-based Failure Onset Annotation

F.1 Annotation Protocol

We prompt the state-of-the-art multimodal model GPT-5.2 to annotate the failure onset of each failed rollout. Each recorded video is provided as input in an offline video question-answering setup, along with the prompt shown below. Note that this setup is less challenging than the online video question-answering used for VLM-based runtime monitoring in Table 6, as GPT-5.2 has access to the entire recorded video rather than a streaming prefix. We use these annotations solely for analyzing the ablation results in Figure 3 and Tables 10 and 13, not for evaluating our main metrics such as TWA.

Prompt for GPT-5.2-based failure onset annotation

System: You are an expert robotics failure annotator.

Goal: Pinpoint the FAILURE START frame index.

Definition (use this strictly):
FAILURE START = the earliest frame t in this window where an irreversible mistake first ap-

pears **or** where the agent's behavior becomes clearly unrecoverable and is not being corrected.

You must:

1. Choose exactly one integer index as `failure_start`.
2. Provide 1–2 pieces of visual evidence referencing specific frames.
3. If you cannot see any decisive change in this window, choose the earliest index where the failure signal is most apparent and set confidence low.

Output format (STRICT JSON only):

```
{
  "failure_start": <int>,
  "confidence": 0.0-1.0,
  "evidence": [
    {"index": <int>, "cue": "what changes / what is wrong"},
    {"index": <int>, "cue": "optional second cue"}
  ],
  "failure_type": "drop | wrong-object | collision |
                   stuck/off-track | goal-violation | other",
  "recoverable_before_start": "yes | no | uncertain",
  "notes": "optional"
}
```

Inputs:
Instruction: {instruction}
Window frames (in order):
[index={t_prev}] {frame}
[index={t_prev+1}] {frame}
...
[index={t_curr}] {frame}

Now produce the JSON.

F.2 Human Agreement Analysis

To validate the GPT-5.2 failure-onset annotations, we conducted a human-agreement study over 76 failed rollouts rated by four PhD-level machine learning researchers. Each rater was shown the recorded video with GPT-5.2's predicted onset marked by a green→red border transition and asked to mark *Agree* or *Disagree*, permitting small temporal offsets. Raters agreed with GPT-5.2 on 90.7% of the 304 ratings; all four raters unanimously agreed on 75.0% of rollouts, and no rollout received unanimous disagreement. This exceeds the $\sim 80\%$ human–LLM agreement typically reported in recent "LLM-as-a-judge" studies with strong LLM judges [75], which itself matches human–human agreement on the same tasks. We therefore treat GPT-5.2 onset annotations as a reliable proxy for human-perceived failure onset in Figure 3 and Tables 10 and 13. The study involved only viewing pre-recorded videos, collected no personal or sensitive information, and was conducted under a minimal-risk setting with voluntary participation.

G Baselines

We provide implementation details for all baselines used in our experiments. Each method operates at a different level of the VLA model: OOD detection-based (Section G.1) and classifier-based (Section G.3) methods operate on VLA action embeddings $h_t \in \mathbb{R}^d$ extracted at each timestep t (see Section 5.1 for extraction details); multi-sampling-based methods (Section G.2) operate on sampled actions $\{a_t^{(n)}\}_{n=1}^N$ drawn from the policy output; and token uncertainty-based methods (Section G.4) operate on per-token likelihoods of autoregressive policies. For embedding-level methods, we denote by $\mathcal{H}_{\text{succ}}$ and $\mathcal{H}_{\text{fail}}$ the sets of all embeddings from successful and failed rollouts in the training set $\mathcal{D}_{\text{train}}$, respectively. Regardless of input modality, each method produces a per-timestep failure score s_t at evaluation time.

G.1 Out-of-Distribution Detection-based Methods

G.1.1 Embedding Distance

Embedding-distance methods use feature-space proximity to compute failure scores directly, without training any neural network. While standard OOD detection models only consider the successful embeddings, we extend these baselines to exploit the same trajectory-level failure supervision available to our method, following recent practice [15]. Specifically, we contrast the distance of the query embedding h_t to both the successful and failed embedding sets: $s_t = d(h_t, \mathcal{H}_{\text{succ}}) - d(h_t, \mathcal{H}_{\text{fail}})$, where $d(\cdot, \cdot)$ is a distance function between a vector and a set. Intuitively, h_t receives a high failure score when it lies far from successful embeddings and close to failed ones.

Mahalanobis distance [55]. We fit a Gaussian to each embedding set and compute the Mahalanobis distance to its mean:

$$d_{\text{Maha}}(h_t, \mathcal{H}) = \sqrt{(h_t - \mu_{\mathcal{H}})^\top \Sigma_{\mathcal{H}}^{-1} (h_t - \mu_{\mathcal{H}})}, \tag{13}$$

where $\mu_{\mathcal{H}}$ and $\Sigma_{\mathcal{H}}$ are the sample mean and covariance of $\mathcal{H}$. This captures deviations in a direction-sensitive manner.

Cosine k-NN [56]. We compute the average cosine distance between h_t and its k nearest neighbors in $\mathcal{H}$:

$$d_{\cos}(h_t, \mathcal{H}) = \frac{1}{k} \sum_{h_j \in \mathcal{N}_k(h_t, \mathcal{H})} \left(1 - \frac{h_t^\top h_j}{\|h_t\| \|h_j\|}\right), \tag{14}$$

where $\mathcal{N}_k(h_t, \mathcal{H})$ denotes the k nearest neighbors of h_t in $\mathcal{H}$ under cosine similarity. Unlike Mahalanobis distance, this is non-parametric and captures local neighborhood structure.

PCA-KMeans [39]. We first project all embeddings in $\mathcal{H}$ onto their top p principal components, then cluster the projected embeddings into K clusters via k-means. The distance to $\mathcal{H}$ is the Euclidean distance from h_t (similarly projected) to the nearest cluster centroid:

$$d_{\text{PCA}}(h_t, \mathcal{H}) = \min_{k=1,\ldots,K} \|P(h_t) - \mu_k\|_2, \tag{15}$$

where $P(\cdot)$ denotes the PCA projection and $\{\mu_k\}$ the cluster centroids.

G.1.2 Learned OOD Detector

Learned OOD detectors train a neural network $f_{\text{OOD}}(\cdot)$ to model the embedding distribution of a given set and output a scalar score. Following the same pairwise supervision strategy, we adapt these methods to the action embedding space by training two separate models [15] $f_{\text{succ}}^{\text{OOD}}$ and $f_{\text{fail}}^{\text{OOD}}$, on $\mathcal{H}_{\text{succ}}$ and $\mathcal{H}_{\text{fail}}$ respectively, and compute: $s_t = f_{\text{succ}}^{\text{OOD}}(h_t) - f_{\text{fail}}^{\text{OOD}}(h_t)$.

RND (Random Network Distillation) [42]. RND uses a fixed randomly initialized target network $\bar{\psi}$ and a trainable predictor ψ, both operating on the action embedding $h_t \in \mathcal{H}$. The per-head score is the predictor–target mismatch:

$$f_{\text{RND}}^{\text{OOD}}(h_t) = \|\psi(h_t) - \bar{\psi}(h_t)\|_2. \tag{16}$$

The predictor is trained to minimize this distance on the corresponding split, so a lower error indicates h_t lies within that distribution, while higher error indicates it does not.

LogpZO [12]. In our implementation, LogpZO is realized via a flow-matching velocity model. The per-head score is an energy-style residual:

$$f_{\text{LogpZO}}^{\text{OOD}}(h_t) = \|h_t + v_\theta(h_t, 0)\|_2^2, \tag{17}$$

where v_θ is the learned velocity field. Higher values indicate greater out-of-distribution risk.

G.2 Multi-Sampling-based Methods

Multi-sampling methods estimate uncertainty by drawing N action samples $\{a_t^{(1)}, \ldots, a_t^{(N)}\}$ from the VLA policy at each timestep and computing a per-step uncertainty score. We use $N = 10$ samples across all methods.

Cluster Entropy [57]. Following the implementation of [15], each timestep t, we cluster the sampled actions using agglomerative hierarchical clustering with a distance threshold ε. Let n_k be the size of cluster k, and $p_k = n_k/N$. The uncertainty score is the Shannon entropy of the induced cluster-size distribution:

$$s_t = -\sum_{k=1}^{K_t} p_k \log p_k, \tag{18}$$

where K_t is the number of clusters formed at timestep t under threshold ε. Higher entropy indicates greater disagreement among sampled actions.

EigenScore [58] measures uncertainty via the spectral dispersion of the sample covariance matrix $\Sigma_t = \mathrm{Cov}(\{a_t^{(n)}\})$:

$$s_t = \log\det(\Sigma_t + \epsilon I), \tag{19}$$

where ϵ is a small regularization constant. Larger determinant implies greater variance across directions. Notably, we compute EigenScore in the action space rather than the intermediate feature space to ensure consistency with other multi-sampling baselines and to directly reflect variability in predicted actions.

STAC (Statistical Measures of Temporal Action Consistency) [17] detects erratic failures by measuring how much a generative policy's action distributions change over time. For two consecutive policy queries at timesteps t and $t+k$, the sampled action chunks $\{a^{(i)}_{t+k:t+h-1|t}\}_{i=1}^N$ and $\{a^{(j)}_{t+k:t+h-1|t+k}\}_{j=1}^N$ share the same $(h-k)$-step overlap window and should agree if the policy is temporally consistent. The failure score is a statistical distance between these two action distributions over the overlap:

$$s_t = D_{\mathrm{stat}}\Big(\{a^{(i)}_{t+k:t+h-1|t}\}, \{a^{(j)}_{t+k:t+h-1|t+k}\}\Big), \tag{20}$$

where D_{stat} is implemented via Maximum Mean Discrepancy (MMD) following the original paper. Large distribution drift across queries indicates erratic policy behavior.

Since OpenVLA does not emit action chunks but produces a single immediate action per timestep, there is no overlap between consecutive policy queries, and STAC is not directly applicable. We adapt STAC to this setting by reinterpreting its core principle–*local temporal consistency of the policy's action distribution*–to operate on single-step actions sampled at adjacent timesteps. Specifically, at each timestep t, we sample N actions from OpenVLA at steps t and $t+1$ and compute the MMD between the two action distributions:

$$s_t = \mathrm{MMD}\Big(\{a_t^{(i)}\}_{i=1}^N, \{a_{t+1}^{(j)}\}_{j=1}^N\Big). \tag{21}$$

A large MMD indicates abrupt change in the sampled action distribution between adjacent steps, analogous to the temporal inconsistency STAC detects in chunk-based policies.

ACE (Action-Chunk Entropy) [16] estimates uncertainty by computing histogram-based entropy over N sampled action chunks. To avoid the curse of dimensionality across the chunk horizon H, ACE treats each prediction step separately: for each step $t+i$, it bins the N sampled actions into an adaptive grid and sums per-step entropies:

$$s_t = \sum_{i=0}^{H-1} \hat{H}\big(a^{(1)}_{t+i|t}, \ldots, a^{(N)}_{t+i|t}\big), \tag{22}$$

where $\hat{H}(\cdot)$ is the Shannon entropy of the histogram over the N sampled actions. High entropy reflects persistent ambiguity in the policy's intended behavior. For OpenVLA, which outputs a single action per step ($H = 1$), the summation reduces to a single-step histogram entropy over the N sampled actions.

G.3 Classifier-based Methods

Classifier-based methods train a neural network on VLA internal embeddings h_t to predict per-timestep failure likelihood from trajectory-level labels $y \in \{0, 1\}$ (with $y = 1$ denoting failure). We

evaluate two SAFE [15] variants that differ in backbone and loss. Both propagate trajectory-level labels across timesteps, overlooking temporal localization and introducing label noise from pre-onset normal actions.

SAFE-MLP uses a two-layer MLP $g(\cdot)$ applied independently at each timestep, treating each embedding as an i.i.d. input. The per-timestep score is passed through a sigmoid, and the failure score at time t is defined as the accumulated sum of scores up to t: $s_t^{\text{MLP}} = \sum_{\tau=1}^{t} \sigma\big(g(h_\tau)\big)$, so that $0 \le s_t^{\text{MLP}} \le t$. The model is trained with an asymmetric L_1 loss that pushes scores up for failed rollouts and down for successful ones:

$$\mathcal{L}_{\text{MLP}} = \sum_{i \in \mathcal{D}_{\text{train}}} \Big[y_i \sum_t (t - s_t) + (1 - y_i) \sum_t s_t \Big]. \tag{23}$$

Note that SAFE-MLP injects temporal structure through a linearly increasing target weight via the cumulative score formulation, but this heuristic is agnostic to when failure actually occurs and treats every later timestep as more failure-indicative regardless of the underlying dynamics.

SAFE-LSTM uses a single-layer LSTM to sequentially process the embedding stream $h_{1:t}$ and project the hidden state into a scalar score, normalized by a sigmoid:$s_t^{\text{LSTM}} = \sigma\big(\text{LSTM}(h_{1:t})\big) \in [0, 1]$. The model is trained with a binary cross-entropy loss applied uniformly to all timesteps:

$$\mathcal{L}_{\text{LSTM}} = -\sum_{i \in \mathcal{D}_{\text{train}}} \sum_t \Big[y_i \log s_t + (1 - y_i) \log(1 - s_t) \Big]. \tag{24}$$

To mitigate the class imbalance between the successful and the failed rollouts, both variants weight the loss contributions by inverse class frequency and apply L_2 regularization on model weights.

G.4 Token Uncertainty-based Methods

Inspired by uncertainty quantification methods for LLM hallucination detection, we adapt token-level probability-based scores to autoregressive VLA models such as OpenVLA, which decode actions as a sequence of discrete tokens. At each timestep t, the VLA generates K action tokens $w_t^{(1)}, \ldots, w_t^{(K)}$, where each token $w_t^{(k)}$ is assigned probability $p_t^{(k)} = p(w_t^{(k)} \mid o_t, w_t^{(1:k-1)})$.

Max / Mean Negative Log-likelihood [60]. We aggregate the negative log-likelihood across the K action tokens:

$$s_t^{\text{max-nll}} = \max_{k=1,\ldots,K} \big(-\log p_t^{(k)} \big), \qquad s_t^{\text{avg-nll}} = (-\frac{1}{K} \sum_{k=1}^{K} \log p_t^{(k)}). \tag{25}$$

The max variant captures the least confident token within an action, highlighting worst-case uncertainty, while the mean provides a smoother estimate of overall uncertainty across tokens.

Max / Mean Token Entropy [59]. Entropy-based scores aggregate the per-token entropy over the action-token vocabulary $\mathcal{V}$. Let $E_t^{(k)} = -\sum_{v \in \mathcal{V}} p_t^{(k)}(v) \log p_t^{(k)}(v)$ denote the entropy at token position k. Then,

$$s_t^{\text{max-ent}} = \max_{k=1,\ldots,K} E_t^{(k)}, \qquad s_t^{\text{avg-ent}} = \frac{1}{K} \sum_{k=1}^{K} E_t^{(k)}. \tag{26}$$

Unlike log-likelihood-based scores, which depend on the sampled tokens, entropy-based scores reflect the full predictive distribution.

G.5 VLM-based Method

We implement a VLM-based runtime monitor using Qwen3-VL-8B-Instruct [65]. At each timestep t, we construct a temporal context consisting of previously sampled frames up to the current frame. The model performs online video question answering (VQA) to predict whether the current frame is in a failure state. A trajectory is classified as a failure if any timestep is predicted as a failure.

Prompt for Qwen3-VL-based runtime failure detection

System: You are a robot monitoring assistant.
Goal: Determine whether the robot is currently in a failure state at the latest frame.
Context: You are given a chronological sequence of frames ending at the current timestep (up to {H} frames).
Task instruction:{instruction}
Question: At the last frame (current timestep), is the robot in a failure state?
Output strictly JSON with keys: failure_now (0 or 1), reason (short string).

H Extended Related Work

Hallucination Detection. Hallucinations refer to outputs generated by Large Language Models (LLMs) or Large Vision-Language Models (LVLMs) that are linguistically plausible but factually unsupported. Detecting hallucinations has become a critical research area due to safety concerns in real-world deployments [76, 77]. A broad line of work approaches this problem through uncertainty-based scoring functions. Logit-based methods use token-level probabilities as uncertainty estimates [60, 59, 78, 79, 80], verbalized methods prompt models to explicitly express their uncertainty in natural language [81, 82], and consistency-based methods measure agreement across multiple sampled responses [57, 58, 83]. More recently, internal state-based approaches leverage hidden representations, suggesting that the internal states of LLMs contain a truthful subspace that separates factual from hallucinated content [84, 85, 86, 87, 88].

Our method extends internal-state probing approaches to the latent embedding space of VLA models. We posit that internal action embeddings encode signals of task success and impending failure beyond what is observable in raw actions or visual inputs.

I Limitations and Future Work

Hide-and-Seek averages across the degree-of-freedom dimension of VLA internal embeddings, which may discard kinematic structure that is informative for certain failure modes, for instance, failures that manifest in specific joints or end-effector configurations. Furthermore, some failure modes are inherently perceptual in nature and may not be fully captured by action-level embeddings alone, hence incorporating visual observations alongside action embeddings could provide complementary failure signals. We hope our framework can be extended to explore kinematic-aware feature aggregation and multimodal failure detectors that jointly leverage visual and action representations.

Beyond failure detection, a promising next step is to close the loop by leveraging Hide-and-Seek's ability to raise alarms before failures fully manifest. Such early warning signals naturally enable several downstream mechanisms: (1) corrective intervention, where the policy is guided away from impending failure before irreversible errors occur [89, 90, 91, 92], (2) recovery, where the robot is restored to a viable state and resumes task execution after failure onset [93, 94], and (3) multi-turn interactive systems, where the agent can proactively request clarification, additional demonstrations, or human assistance when failure risk becomes high [95, 96, 97]. Integrating Hide-and-Seek as a front-end runtime monitor for corrective, recovery, and interactive decision-making systems is a promising direction toward robust and self-correcting VLA deployment.

J Broader Impacts

This work contributes to the reliable deployment of Vision-Language-Action models in real-world robotic systems by enabling timely detection of execution failures during runtime. Reliable failure detection can reduce the risk of physical harm, property damage, and unintended interactions when VLAs are deployed in human-shared environments. For instance, in kitchen or household manipulation scenarios, Hide-and-Seek can detect when a robot misgrasps a fragile object or moves off-trajectory toward nearby humans, enabling the system to pause, request human assistance, or trigger recovery procedures before harm occurs.

However, failure detectors are not infallible. False negatives may allow failures to go undetected, while false positives may trigger unnecessary interventions that disrupt task execution or degrade user experience. We therefore position Hide-and-Seek as a practical monitoring component within a broader system, which may include complementary safeguards such as hardware constraints or higher-level supervision, rather than a standalone guarantee of reliable behavior.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction discuss in detail the studied problem and the contributions of our paper.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Limitation is discussed in Section I.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

Justification: Our paper does not include theoretical results.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: We disclose the experimental details in Sections B and E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

Answer: [Yes]

Justification: We use publicly available benchmarks and release our full implementation through an anonymous project page.

Guidelines:

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

6. **Experimental setting/details**

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

Answer: [Yes]

Justification: We disclose the experimental details in Sections B and E.

Guidelines:

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

7. **Experiment statistical significance**

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

Answer: [Yes]

Justification: We report the mean and standard deviation over three random seeds in Section 5.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: We report the computing resources used in Section B.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: We reviewed the NeurIPS Code of Ethics and confirmed that our work does not deviate from it.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: We discuss the broader impact of our work in Section J.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: Our work focuses on failure detection for VLA models and does not involve artifacts that pose a high risk of misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: We cite relevant works for the resource we use for the experiments.

    Guidelines:

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

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [N/A]

    Justification: This paper does not release new assets.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects. The label-agreement analysis involved four volunteer collaborators rating a machine learning artifact (model-generated failure-onset annotations) on pre-recorded robot rollout videos; no data was collected about the annotators themselves, and the activity does not meet the regulatory definition of human subjects research. For transparency, we still document the annotation instructions in Section F and note that participation was voluntary and uncompensated.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: Our human-agreement analysis involved four volunteer annotators rating the alignment of model-generated failure-onset labels with their own judgments on pre-recorded robot rollout videos. The study collected only label-agreement responses about a machine learning artifact (the GPT-5.2 annotations), not data about the annotators themselves; no personally identifiable, sensitive, or private information was collected. Annotators provided informed consent, were not compensated, could withdraw at any time, and the study presented no foreseeable risks beyond those of everyday computer use. Under applicable institutional guidelines, this activity does not meet the regulatory definition of human subjects research and therefore did not require IRB review or an exemption determination. The study was conducted in accordance with the NeurIPS Code of Ethics.

Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: LLMs were used only for grammar correction and editing, and do not affect the core methodology. GPT-5.2 annotations are used only for supporting analyses, not in the core method.

    Guidelines:

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