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]

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 (Kim et al., 2025; Black et al., 2024; Intelligence et al., 2025a; Bjorck et al., 2025; Wu et al., 2026; Zhang et al., 2026; Torne et al., 2026; Sridhar et al., 2026) enable robot policies to follow natural language instructions and learn from heterogeneous, multi-modal demonstrations, showing promising generalization to novel environments and embodiments (Xu et al., 2025b). Despite these advances, VLAs remain susceptible to diverse execution failures that undermine reliability and increase the cost of real-world deployment (Visinsky et al., 1994; Rahman et al., 2021). 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 (Xu et al., 2025a; Ali et al., 2025; Ramrakhya et al., 2025).

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 (Gu et al., 2025) 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 (Römer et al., 2025; Agia et al., 2024; Jang et al., 2026) or external VLM judges (Yuan et al., 2026; Duan et al., 2025; Qi et al., 2026) incur substantial inference overhead that precludes real-time deployment. Most prior methods are additionally designed for specialist policies on fixed tasks (Xu et al., 2025a; Farid et al., 2022; Liu et al., 2024a; Seo et al., 2025), 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.

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

[1]University of Wisconsin–Madison [2]Georgia Institute of Technology. Correspondence to: Seongheon Park <seongheon_park@cs.wisc.edu>, Sharon Li <sharonli@cs.wisc.edu>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

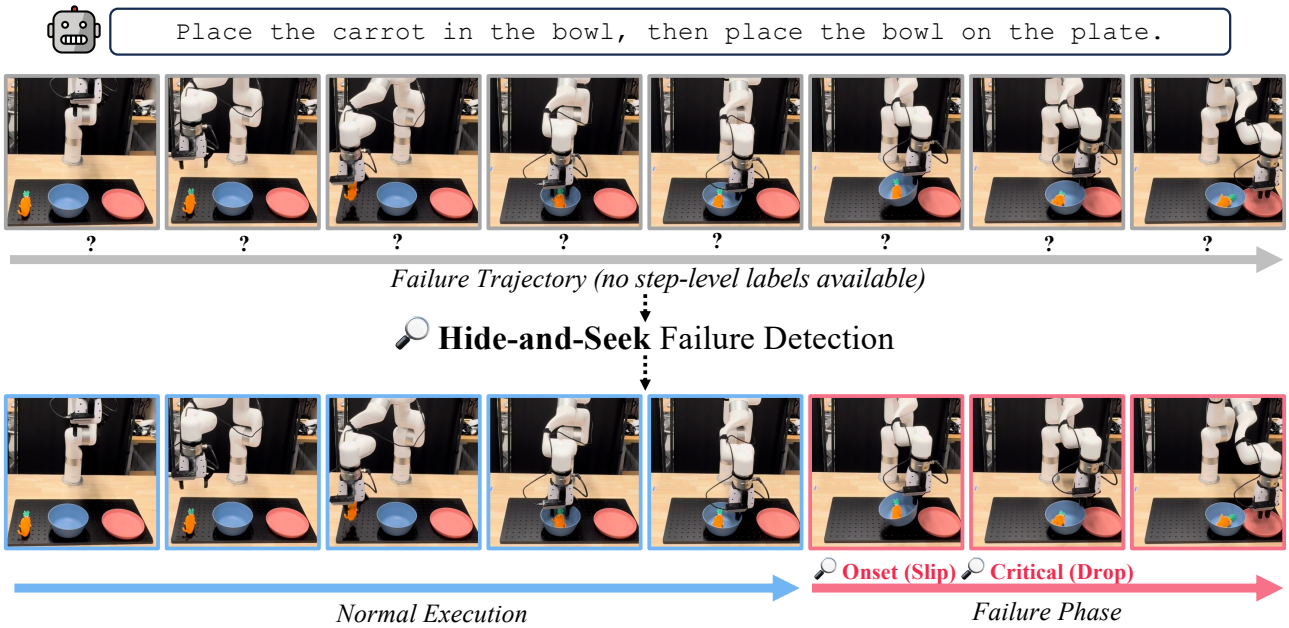


Figure 1. **Hide-and-Seek Failure Detection.** Failure trajectories contain a substantial amount of normal actions before failure onset, yet only a trajectory-level label is available during training, leaving the temporal structure entirely unknown (top). From this coarse supervision, Hide-and-Seek discovers 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).

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 (Liu et al., 2023a) and VLABench (Zhang et al., 2025), as well as on a real-world robotic platform, using three representative VLA policies spanning both autoregressive and flow-matching-based paradigms: OpenVLA (Kim et al., 2025), π_0 (Black et al., 2024), and $\pi_{0.5}$ (Intelligence et al., 2025b). 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 (Kim et al., 2025; Zheng et al., 2025; Qu et al., 2025; Cen et al., 2025; Pertsch et al., 2025; Liu et al., 2026; Wang et al., 2025), producing actions token-by-token in a manner analogous to language modeling (Radford et al., 2019; Liu et al., 2023b). 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 (Black et al., 2024; Intelligence et al., 2025b;a; Bjorck et al., 2025; Barreiros et al., 2025; Ye et al., 2026; Torne et al., 2026). Our method is architecture-agnostic and compatible with foundational VLA policies spanning both paradigms: the autoregressive model OpenVLA (Kim et al., 2025) and the flow-matching-based models π_0 (Black et al., 2024) and $\pi_{0.5}$ (Intelligence et al., 2025b). 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 (Visinsky et al., 1994; Rahman et al., 2021; Antonante et al., 2023). Existing approaches fall into four categories: (1) OOD detection-based methods (Liu et al., 2024c; Xu et al., 2025a; Zhou et al., 2026; Sinha et al., 2024; He et al., 2024) treat successful executions as in-distribution and flag deviations as failures, yet struggle to generalize to unseen distributions. (2) VLM-based methods (Yuan et al., 2026; Duan et al., 2025; Qi et al., 2026; Agia et al., 2024) 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 (Römer et al., 2025; Agia et al., 2024; Jang et al., 2026) quantify uncertainty per action via ensemble sampling, but incur prohibitive inference costs that limit practical deployment. (4) Classifier-based methods (Barreiros et al., 2025; Seo et al., 2025; Gu et al., 2025; Liu et al., 2024a; Farid et al., 2022) train failure classifiers by either collecting costly step-level annotations 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 (Jiang et al., 2023; Zhang et al., 2021). Often framed as weakly supervised learning, it has been successfully applied to object detection (Tang et al., 2017; Wan et al., 2019), video temporal action localization (Lee et al., 2021; Tang et al., 2023), and anomaly detection (Sultani et al., 2018; Pang et al., 2023). For example, multiple instance learning (Carbonneau et al., 2018) 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 $\tau_{\leq t} = (h_1, h_2, \ldots, h_t)$ 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 $\tau_{\leq t} = (h_1, h_2, \ldots, h_t)$, the detector produces a binary decision as follows:*

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

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.

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

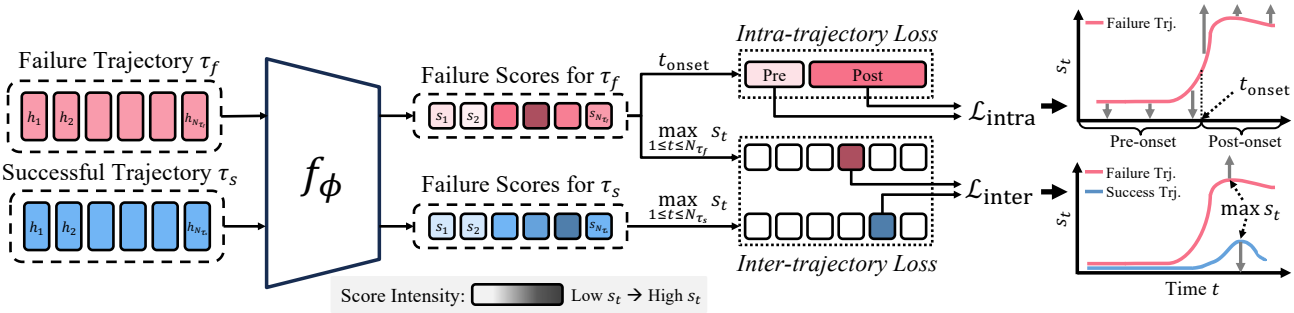


Figure 2. **Overall framework.** Given a failure-success trajectory pair (τ_f, τ_s), the detector f_ϕ produces per-step failure scores s_t for both failure and successful trajectories. $\mathcal{L}_{\text{inter}}$ (Eq. 2) enforces that the most failure-indicative action in the failure trajectory τ_f 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.

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 E.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:

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

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 becomes less likely after failure onset (Ross et al., 2011), 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:

$$\bar{s}_{\text{post}}(\tau_f) = \frac{1}{N_{\tau_f} - t_{\text{onset}} + 1} \sum_{t\ge t_{\text{onset}}} s_t,$$
$$\bar{s}_{\text{pre}}(\tau_f) = \frac{1}{t_{\text{onset}} - 1} \sum_{t< t_{\text{onset}}} s_t,$$
$$\mathcal{L}_{\text{intra}} = \frac{1}{|\mathcal{D}_f|} \sum_{\tau_f\in\mathcal{D}_f} \max(0,\ m_o - \bar{s}_{\text{post}}(\tau_f) + \bar{s}_{\text{pre}}(\tau_f)) \,. \tag{3}$$

where $m_o > 0$ is a margin hyperparameter. In Section D.1, we show that t_{onset} closely approximates the annotated failure onset and achieves comparable performance when used

[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 2 and Section E.3.

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) (Diquigiovanni et al., 2021), 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 (Xu et al., 2025a; Gu et al., 2025) 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 (Zhao et al., 2023), 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 C.

Benchmarks and models. We evaluate *multi-task* failure detection on two simulation benchmarks, LIBERO-10 (Liu et al., 2023a) and VLABench (Zhang et al., 2025), and a real-world UFactory xArm 6 platform, using representative VLA policies including OpenVLA (Kim et al., 2025), π_0 (Black et al., 2024), and $\pi_{0.5}$ (Intelligence et al., 2025b). 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 F.

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)** (Römer et al., 2025) 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 C.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 (Lee et al., 2018), Cosine k-NN (Sun et al., 2022), PCA-KMeans (Liu et al., 2024c), RND (He et al., 2024), and LogpZO (Xu et al., 2025a). (2) **Multi-sampling-based methods**[2] assess uncertainty through repeated action sampling: Cluster Entropy (Kuhn et al., 2023), EigenScore (Chen et al., 2024), STAC (Agia et al., 2024), and ACE (Römer et al., 2025). (3) **Classifier-based methods** train a failure classifier on frozen VLA representations: SAFE (Gu et al., 2025). For

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

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%)								
			Seen Tasks			Unseen Tasks		
	Method		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%)								
			Seen Tasks			Unseen Tasks		
	Method		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 H.2).

OpenVLA, we additionally evaluate (4) **Token uncertainty-based methods**, which estimate failure likelihood from per-action token probabilities: Entropy (Malinin & Gales, 2021), Negative log-likelihood (NLL) (Ren et al., 2023). Details about each baseline are provided in Section H.

5.2. Main Experiments

Simulation environment. Table 1 report failure detection performance on LIBERO-10 using OpenVLA and π_0. 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 +**2.9**% bACC and +**3.0**%

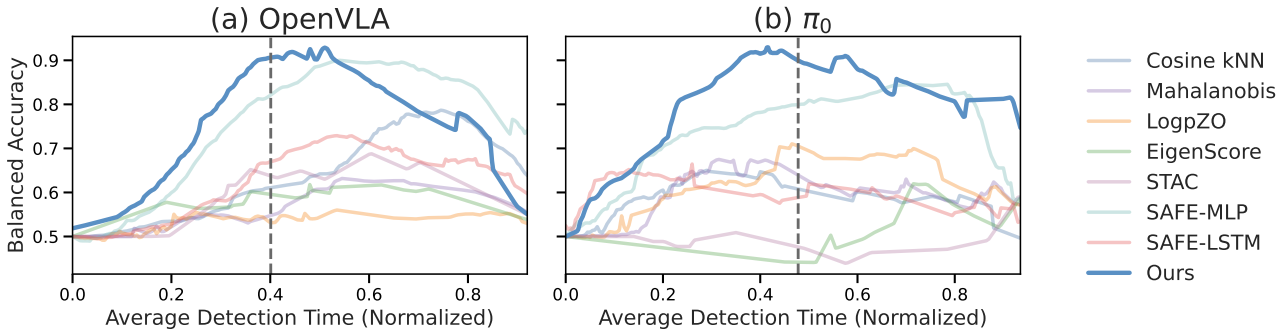


Figure 3. Detection accuracy–timeliness tradeoff on LIBERO-10 with OpenVLA and π_0. Gray dashed line indicates the average normalized failure onset time across failure trajectories.

Table 2. 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 3. 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 4. Comparison with VLM-based runtime monitor.

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

TWA on seen tasks, and +**5.9**% bACC and +**7.3**% TWA on unseen tasks with OpenVLA. Results on VLABench and real-robot experiments using $\pi_{0.5}$ are provided in **Section A**. Qualitative results are provided in **Section B.2**.

5.3. Ablation Studies

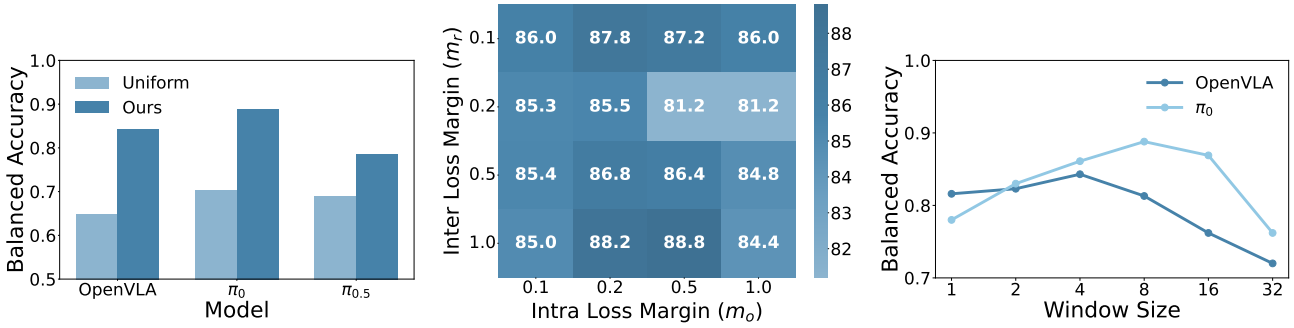
Detection accuracy–timeliness trade-off. A reliable failure detector must balance accuracy and promptness. In our framework, the user-specified α controls this trade-off: stricter α yields more conservative thresholds with delayed detection, while looser α enables earlier detection at the cost of more 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 (Singh et al., 2025) on recorded videos, shown as vertical dashed lines (see Section G.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 E.2.

Detector architecture. We compare our LSTM-based (Graves, 2012) detector[3] against three alternatives in Table 2: an MLP, a GRU (Chung et al., 2014), and a single-layer Transformer (Vaswani et al., 2017). 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.

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

Loss component analysis. We ablate the contribution of each training objective in Table 3. 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-



(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.

VL-8B-Instruct (Bai et al., 2025) 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 as a failure if at least one frame is judged as a failure. As shown in Table 4, 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 (Gu et al., 2025), 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 E.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, and (3) layer-wise effects are provided in Section D.

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 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, National Science Foundation under awards IIS-2237037 and IIS2331669, 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

Agia, C., Sinha, R., Yang, J., Cao, Z.-a., Antonova, R., Pavone, M., and Bohg, J. Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress. In *CoRL*, 2024.

Ali, Y., Patlin, G., Kothuri, K., Irshad, M. Z., Liang, W., and Kira, Z. Eve: A generator-verifier system for generative policies. *arXiv preprint arXiv:2512.21430*, 2025.

Angelopoulos, A. N. and Bates, S. Conformal prediction: A gentle introduction. *Foundations and Trends in Machine Learning*, 2023.

Antonante, P., Nilsen, H. G., and Carlone, L. Monitoring of perception systems: Deterministic, probabilistic, and learning-based fault detection and identification. *Artificial Intelligence*, 2023.

Azaria, A. and Mitchell, T. The internal state of an LLM knows when it's lying. In *EMNLP Findings*, 2023.

Bai, S., Cai, Y., Chen, R., Chen, K., Chen, X., Cheng, Z., Deng, L., Ding, W., Gao, C., Ge, C., et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Barreiros, J., Beaulieu, A., Bhat, A., Cory, R., Cousineau, E., Dai, H., Fang, C.-H., Hashimoto, K., Irshad, M. Z., Itkina, M., et al. A careful examination of large behavior models for multitask dexterous manipulation. *arXiv preprint arXiv:2507.05331*, 2025.

Bjorck, J., Castañeda, F., Cherniadev, N., Da, X., Ding, R., Fan, L., Fang, Y., Fox, D., Hu, F., Huang, S., et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Black, K., Brown, N., Driess, D., Esmail, A., Equi, M., Finn, C., Fusai, N., Groom, L., Hausman, K., Ichter, B., et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Carbonneau, M.-A., Cheplygina, V., Granger, E., and Gagnon, G. Multiple instance learning: A survey of problem characteristics and applications. *Pattern recognition*, 2018.

Cen, J., Huang, S., Yuan, Y., Li, K., Yuan, H., Yu, C., Jiang, Y., Guo, J., Li, X., Luo, H., et al. Rynnvla-002: A unified vision-language-action and world model. *arXiv preprint arXiv:2511.17502*, 2025.

Chen, C., Liu, K., Chen, Z., Gu, Y., Wu, Y., Tao, M., Fu, Z., and Ye, J. Inside: Llms' internal states retain the power of hallucination detection. In *ICLR*, 2024.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Diquigiovanni, J., Fontana, M., and Vantini, S. The importance of being a band: Finite-sample exact distribution-free prediction sets for functional data. *arXiv preprint arXiv:2102.06746*, 2021.

Du, X., Xiao, C., and Li, Y. Haloscope: Harnessing unlabeled llm generations for hallucination detection. In *NeurIPS*, 2024.

Duan, J., Pumacay, W., Kumar, N., Wang, Y. R., Tian, S., Yuan, W., Krishna, R., Fox, D., Mandlekar, A., and Guo, Y. Aha: A vision-language-model for detecting and reasoning over failures in robotic manipulation. In *ICLR*, 2025.

Farid, A., Snyder, D., Ren, A. Z., and Majumdar, A. Failure prediction with statistical guarantees for vision-based robot control. In *RSS*, 2022.

Graves, A. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, 2012.

Groot, T. and Valdenegro-Toro, M. 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.

Gu, Q., Ju, Y., Sun, S., Gilitschenski, I., Nishimura, H., Itkina, M., and Shkurti, F. Safe: Multitask failure detection for vision-language-action models. In *NeurIPS*, 2025.

He, N., Li, S., Li, Z., LIU, Y., and He, Y. Rediffuser: Reliable decision-making using a diffuser with confidence estimation. In *ICML*, 2024.

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

Intelligence, P., Amin, A., Aniceto, R., Balakrishna, A., Black, K., Conley, K., Connors, G., Darpinian, J., Dhabalia, K., DiCarlo, J., et al. $\pi_{0.6}$: a vla that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025a.

Intelligence, P., Black, K., Brown, N., Darpinian, J., Dhabalia, K., Driess, D., Esmail, A., Equi, M., Finn, C., Fusai, N., et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025b.

Jang, S., Kim, D., Kim, C., Kim, Y., and Shin, J. Verifier-free test-time sampling for vision language action models. In *ICLR*, 2026.

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., and Fung, P. Survey of hallucination in natural language generation. *ACM computing surveys*, 2023.

Jiang, M., Hou, C., Zheng, A., Hu, X., Han, S., Huang, H., He, X., Yu, P. S., and Zhao, Y. Weakly supervised anomaly detection: A survey. *arXiv preprint arXiv:2302.04549*, 2023.

Jiang, Z., Chen, J., Zhu, B., Luo, T., Shen, Y., and Yang, X. Devils in middle layers of large vision-language models: Interpreting, detecting and mitigating object hallucinations via attention lens. In *CVPR*, 2025.

Kim, M. J., Pertsch, K., Karamcheti, S., Xiao, T., Balakrishna, A., Nair, S., Rafailov, R., Foster, E., Lam, G., Sanketi, P., et al. Openvla: An open-source vision-language-action model. In *CoRL*, 2025.

Kuhn, L., Gal, Y., and Farquhar, S. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *ICLR*, 2023.

Lee, K., Lee, K., Lee, H., and Shin, J. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *NeurIPS*, 2018.

Lee, P., Wang, J., Lu, Y., and Byun, H. Weakly-supervised temporal action localization by uncertainty modeling. In *AAAI*, 2021.

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

Liu, B., Zhu, Y., Gao, C., Feng, Y., Liu, Q., Zhu, Y., and Stone, P. Libero: Benchmarking knowledge transfer for lifelong robot learning. In *NeurIPS*, 2023a.

Liu, H., Li, C., Wu, Q., and Lee, Y. J. Visual instruction tuning. In *NeurIPS*, 2023b.

Liu, H., Dass, S., Martín-Martín, R., and Zhu, Y. Model-based runtime monitoring with interactive imitation learning. In *ICRA*, 2024a.

Liu, H., Xue, W., Chen, Y., Chen, D., Zhao, X., Wang, K., Hou, L., Li, R., and Peng, W. A survey on hallucination in large vision-language models. *arXiv preprint arXiv:2402.00253*, 2024b.

Liu, H., Zhang, Y., Betala, V., Zhang, E., Liu, J., Ding, C., and Zhu, Y. Multi-task interactive robot fleet learning with visual world models. In *CoRL*, 2024c.

Liu, Y., Zhang, S., Dong, Z., Ye, B., Yuan, T., Yu, X., Yin, L., Lu, C., Shi, J., Yu, L. J.-T., Zheng, L., Gong, J., Jiang, T., Qiu, X., and Zhao, H. FASTer: Toward powerful and efficient autoregressive vision–language–action models with learnable action tokenizer and block-wise decoding. In *ICLR*, 2026.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *ICLR*, 2019.

Malinin, A. and Gales, M. Uncertainty estimation in autoregressive structured prediction. In *ICLR*, 2021.

Niu, M., Haddadi, H., and Pang, G. Robust hallucination detection in llms via adaptive token selection. In *NeurIPS*, 2025.

Pang, G., Shen, C., Jin, H., and Van Den Hengel, A. Deep weakly-supervised anomaly detection. In *KDD*, 2023.

Park, S., Du, X., Yeh, M.-H., Wang, H., and Li, Y. Steer llm latents for hallucination detection. In *ICML*, 2025.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

Pertsch, K., Stachowicz, K., Ichter, B., Driess, D., Nair, S., Vuong, Q., Mees, O., Finn, C., and Levine, S. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Qi, C., Wang, X., Yong, S., Sheng, S., Mao, H., Srinivasan, S., Nambi, M., Zhang, A., and Dattatreya, Y. Self-refining vision language model for robotic failure detection and reasoning. In *ICLR*, 2026.

Qu, D., Song, H., Chen, Q., Yao, Y., Ye, X., Ding, Y., Wang, Z., Gu, J., Zhao, B., Wang, D., et al. Spatialvla: Exploring spatial representations for visual-language-action model. In *RSS*, 2025.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

Rahman, Q. M., Corke, P., and Dayoub, F. Run-time monitoring of machine learning for robotic perception: A survey of emerging trends. *IEEE Access*, 2021.

Ramrakhya, R., Chang, M., Puig, X., Desai, R., Kira, Z., and Mottaghi, R. Grounding multimodal llms to embodied agents that ask for help with reinforcement learning. *arXiv preprint arXiv:2504.00907*, 2025.

Ren, J., Luo, J., Zhao, Y., Krishna, K., Saleh, M., Lakshminarayanan, B., and Liu, P. J. Out-of-distribution detection and selective generation for conditional language models. In *ICLR*, 2023.

Römer, R., Kobras, A., Worbis, L., and Schoellig, A. P. Failure prediction at runtime for generative robot policies. In *NeurIPS*, 2025.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

Seo, J., Nakamura, K., and Bajcsy, A. Uncertainty-aware latent safety filters for avoiding out-of-distribution failures. In *CoRL*, 2025.

Singh, A., Fry, A., Perelman, A., Tart, A., Ganesh, A., El-Kishky, A., McLaughlin, A., Low, A., Ostrow, A., Ananthram, A., et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Sinha, R., Elhafsi, A., Agia, C., Foutter, M., Schmerling, E., and Pavone, M. Real-time anomaly detection and reactive planning with large language models. In *RSS*, 2024.

Skean, O., Arefin, M. R., Zhao, D., Patel, N., Naghiyev, J., LeCun, Y., and Shwartz-Ziv, R. Layer by layer: Uncovering hidden representations in language models. In *ICML*, 2025.

Sridhar, A., Pan, J., Sharma, S., and Finn, C. Memer: Scaling up memory for robot control via experience retrieval. In *ICLR*, 2026.

Sultani, W., Chen, C., and Shah, M. Real-world anomaly detection in surveillance videos. In *CVPR*, 2018.

Sun, Y., Ming, Y., Zhu, X., and Li, Y. Out-of-distribution detection with deep nearest neighbors. In *ICML*, 2022.

Tang, P., Wang, X., Bai, X., and Liu, W. Multiple instance detection network with online instance classifier refinement. In *CVPR*, 2017.

Tang, X., Fan, J., Luo, C., Zhang, Z., Zhang, M., and Yang, Z. Ddg-net: Discriminability-driven graph network for weakly-supervised temporal action localization. In *ICCV*, 2023.

Torne, M., Pertsch, K., Walke, H., Vedder, K., Nair, S., Ichter, B., Ren, A. Z., Wang, H., Tang, J., Stachowicz, K., et al. Mem: Multi-scale embodied memory for vision language action models. *arXiv preprint arXiv:2603.03596*, 2026.

Van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research*, 2008.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *NeurIPS*, 2017.

Visinsky, M. L., Cavallaro, J. R., and Walker, I. D. Robotic fault detection and fault tolerance: A survey. *Reliability Engineering & System Safety*, 1994.

Wan, F., Liu, C., Ke, W., Ji, X., Jiao, J., and Ye, Q. C-mil: Continuation multiple instance learning for weakly supervised object detection. In *CVPR*, 2019.

Wang, Y., Li, X., Wang, W., Zhang, J., Li, Y., Chen, Y., Wang, X., and Zhang, Z. Unified vision-language-action model. *arXiv preprint arXiv:2506.19850*, 2025.

Wu, X.-M., Fan, B., Liao, K., Jiang, J.-J., Yang, R., Luo, Y., Wu, Z., Zheng, W.-S., and Loy, C. C. Vlanext: Recipes for building strong vla models. *arXiv preprint arXiv:2602.18532*, 2026.

Xu, C., Nguyen, T. K., Dixon, E., Rodriguez, C., Miller, P., Lee, R., Shah, P., Ambrus, R., Nishimura, H., and Itkina, M. Can we detect failures without failure data? uncertainty-aware runtime failure detection for imitation learning policies. In *RSS*, 2025a.

Xu, C., Zhang, S., Liu, Y., Sun, B., Chen, W., Xu, B., Liu, Q., Wang, J., Wang, S., Luo, S., et al. An anatomy of vision-language-action models: From modules to milestones and challenges. *arXiv preprint arXiv:2512.11362*, 2025b.

Ye, S., Ge, Y., Zheng, K., Gao, S., Yu, S., Kurian, G., Indupuru, S., Tan, Y. L., Zhu, C., Xiang, J., et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

Yuan, J., Wu, Y., and Bajcsy, A. When to act, ask, or learn: Uncertainty-aware policy steering. *arXiv preprint arXiv:2602.22474*, 2026.

Zhang, D., Han, J., Cheng, G., and Yang, M.-H. Weakly supervised object localization and detection: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

Zhang, J., Chen, X., Wang, Q., Li, M., Guo, Y., Hu, Y., Zhang, J., Bai, S., Lin, J., and Chen, J. Vlm4vla: Revisiting vision-language-models in vision-language-action models. *arXiv preprint arXiv:2601.03309*, 2026.

Zhang, R., Zhang, H., and Zheng, Z. Vl-uncertainty: Detecting hallucination in large vision-language model via uncertainty estimation. *arXiv preprint arXiv:2411.11919*, 2024.

Zhang, S., Xu, Z., Liu, P., Yu, X., Li, Y., Gao, Q., Fei, Z., Yin, Z., Wu, Z., Jiang, Y.-G., et al. Vlabench: A large-scale benchmark for language-conditioned robotics manipulation with long-horizon reasoning tasks. In *ICCV*, 2025.

Zhao, T. Z., Kumar, V., Levine, S., and Finn, C. Learning fine-grained bimanual manipulation with low-cost hardware. In *RSS*, 2023.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *NeurIPS*, 2023.

Zheng, R., Liang, Y., Huang, S., Gao, J., III, H. D., Kolobov, A., Huang, F., and Yang, J. TraceVLA: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies. In *ICLR*, 2025.

Zhou, S., Zhu, B., Yang, J., Zhao, X., Chen, J., and Jiang, Y.-G. Rc-nf: Robot-conditioned normalizing flow for real-time anomaly detection in robotic manipulation. In *CVPR*, 2026.

Zhou, Y., Cui, C., Yoon, J., Zhang, L., Deng, Z., Finn, C., Bansal, M., and Yao, H. Analyzing and mitigating object hallucination in large vision-language models. In *ICLR*, 2024.

A. Main Results

A.1. Experiment Results on VLABench

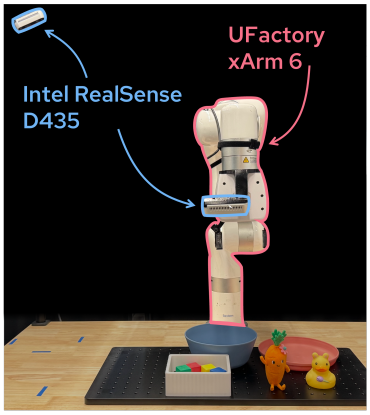
Table 5. Comparison of failure detection performance with $\pi_{0.5}$ on VLABench.

Policy: $\pi_{0.5}$ (Success Rate: 40.3%)								
			Seen Tasks			Unseen Tasks		
	Method		bACC↑	wACC↑	TWA↑	bACC↑	wACC↑	TWA↑
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*	$0.535^{\pm 0.023}$	$0.384^{\pm 0.044}$	$0.485^{\pm 0.035}$	$\underline{0.704}^{\pm 0.072}$	$\underline{0.676}^{\pm 0.095}$	$\underline{0.584}^{\pm 0.088}$
	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*	$0.629^{\pm 0.050}$	$0.656^{\pm 0.069}$	$\underline{0.602}^{\pm 0.029}$	$0.526^{\pm 0.055}$	$0.506^{\pm 0.067}$	$0.490^{\pm 0.050}$
	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*	$\underline{0.788}^{\pm 0.111}$	$\underline{0.766}^{\pm 0.106}$	$0.581^{\pm 0.084}$	$0.641^{\pm 0.029}$	$0.634^{\pm 0.084}$	$0.500^{\pm 0.038}$
	Ours		$\mathbf{0.856}^{\pm 0.075}$	$\mathbf{0.827}^{\pm 0.098}$	$\mathbf{0.662}^{\pm 0.100}$	$\mathbf{0.713}^{\pm 0.035}$	$\mathbf{0.709}^{\pm 0.023}$	$\mathbf{0.608}^{\pm 0.017}$

We further evaluate Hide-and-Seek on VLABench with the $\pi_{0.5}$ policy, which exhibits a substantially lower success rate (40.3%) than the LIBERO-10 setting. As shown in Table 5, our method consistently achieves the best performance across all metrics on both seen and unseen tasks, 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. Notably, the performance gap over OOD detection and multi-sampling baselines widens further in this lower-success-rate regime, where failures are more diverse and harder to characterize without explicit failure supervision. These results demonstrate that Hide-and-Seek generalizes robustly beyond LIBERO-10 to a distinct benchmark and policy, confirming that adaptively localizing failure-indicative moments remains effective even when the underlying policy is weaker.

A.2. Experiment Results on Real Robot

Table 6. Real-world robot setup (left) and failure detection performance with $\pi_{0.5}$ (right).



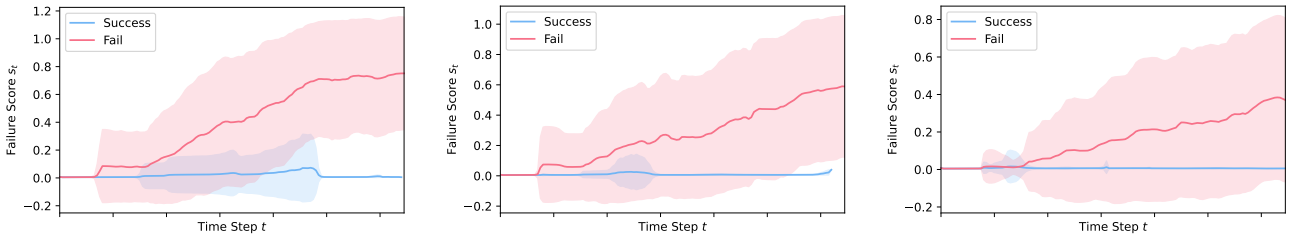
	CUBE (Success Rate: 61.2%)				KITCHEN (Success Rate: 63.7%)			
	Seen		Unseen		Seen		Unseen	
Method	**bACC**	**TWA**	**bACC**	**TWA**	**bACC**	**TWA**	**bACC**	**TWA**
Mahalanobis	0.683	0.652	0.681	0.623	0.779	0.754	0.630	0.608
Cosine k-NN	0.674	0.617	0.658	0.647	0.693	0.612	0.644	0.592
PCA KMeans	0.525	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	<u>0.895</u>	<u>0.811</u>	0.837	<u>0.794</u>
SAFE-LSTM	0.723	0.597	0.714	0.623	0.846	0.780	0.595	0.577
SAFE-MLP	**0.992**	<u>0.847</u>	<u>0.797</u>	<u>0.650</u>	0.784	0.748	<u>0.878</u>	0.768
Ours	<u>0.966</u>	**0.852**	**0.914**	**0.800**	**0.968**	**0.863**	**0.972**	**0.876**

Table 6 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 F.2.

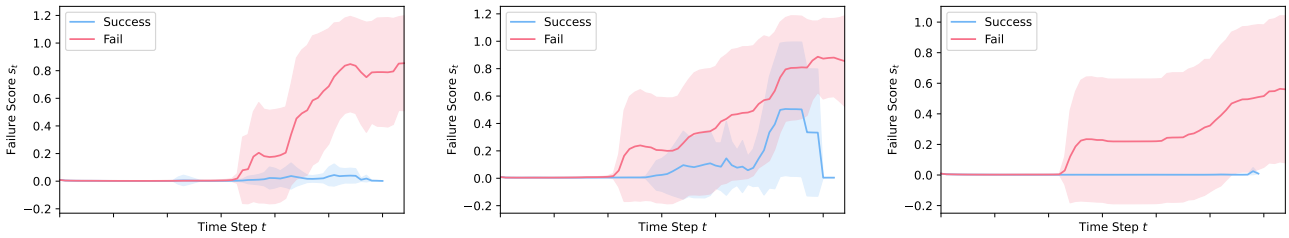
B. Failure Score Visualization

B.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.**

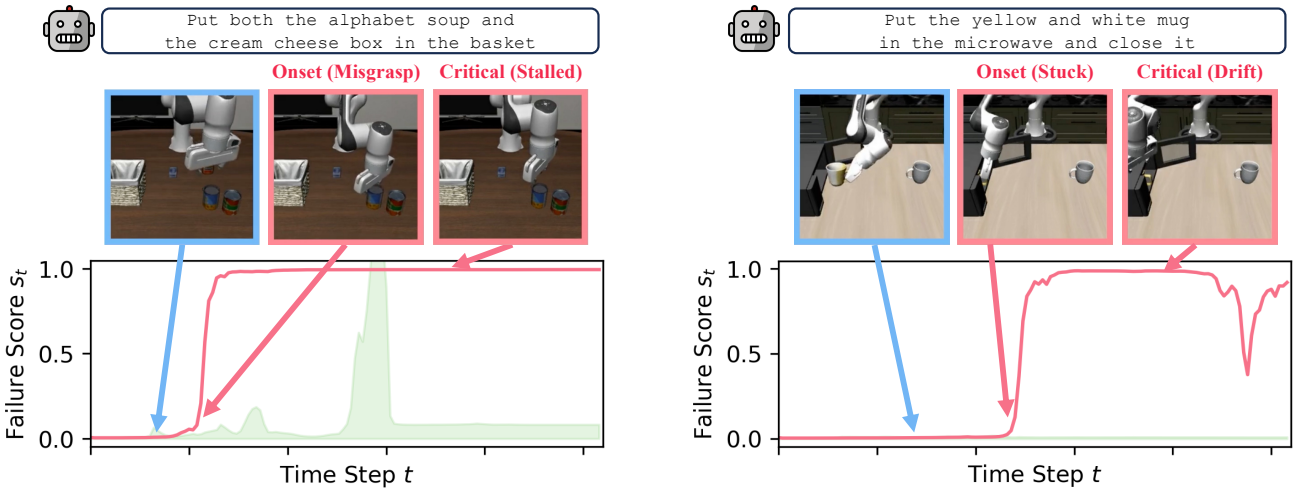
We visualize the average per-timestep failure score s_t across episodes, computed after training the detector, for the training set and the seen/unseen evaluation splits. Results are shown for OpenVLA (Figure 5) and π_0 (Figure 6) on LIBERO-10. Across all splits, the failure score remains consistently 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.

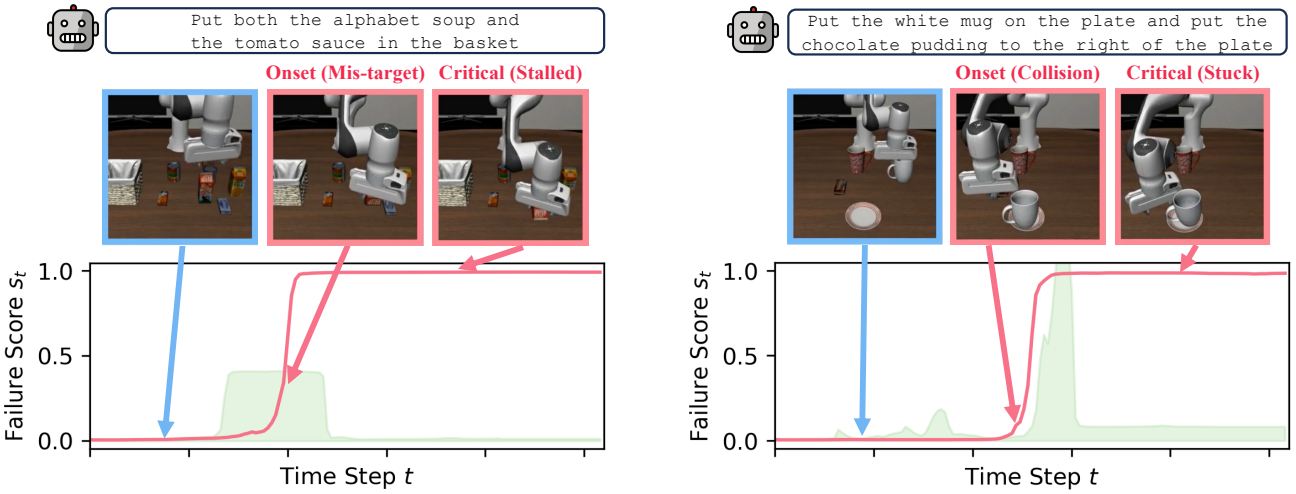
B.2. Qualitative Results

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.

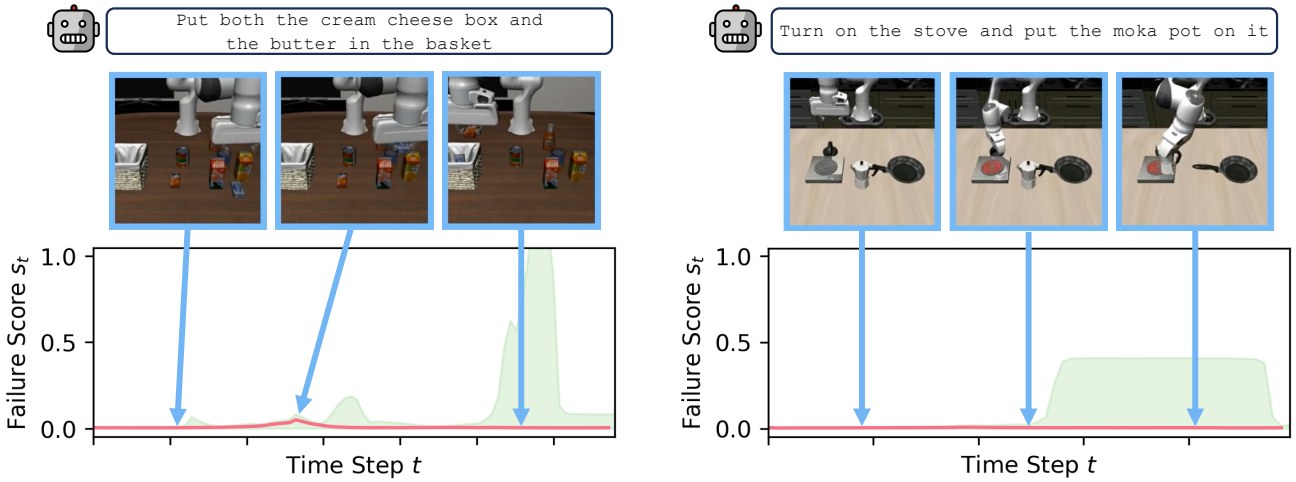
We visualize qualitative examples from OpenVLA (Figure 7) and π_0 (Figure 8) on LIBERO-10 tasks. A failure is declared at the earliest timestep t such that the failure score s_t (red curve) exceeds the threshold ζ_t (green region). The CP threshold ζ_t adapts to score variability across the calibration 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.

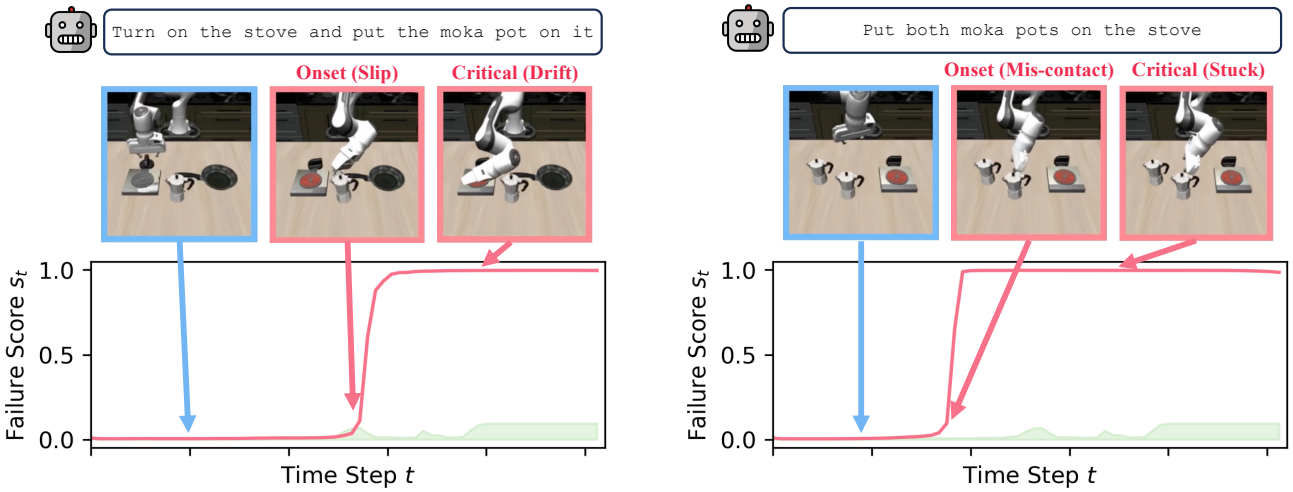


(b) **Failure Trajectories.** (left) Robot grasps the wrong object, drops it, and stalls. (right) Robot collides with the placed mug and gets stuck.

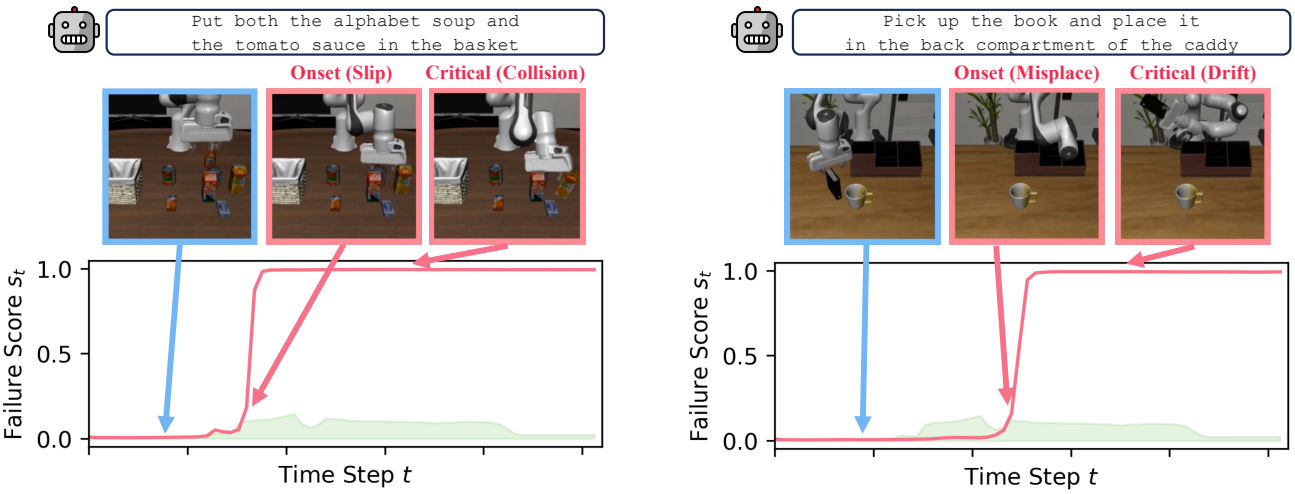


(c) **Successful Trajectories.** The failure score remains consistently low throughout the trajectory.

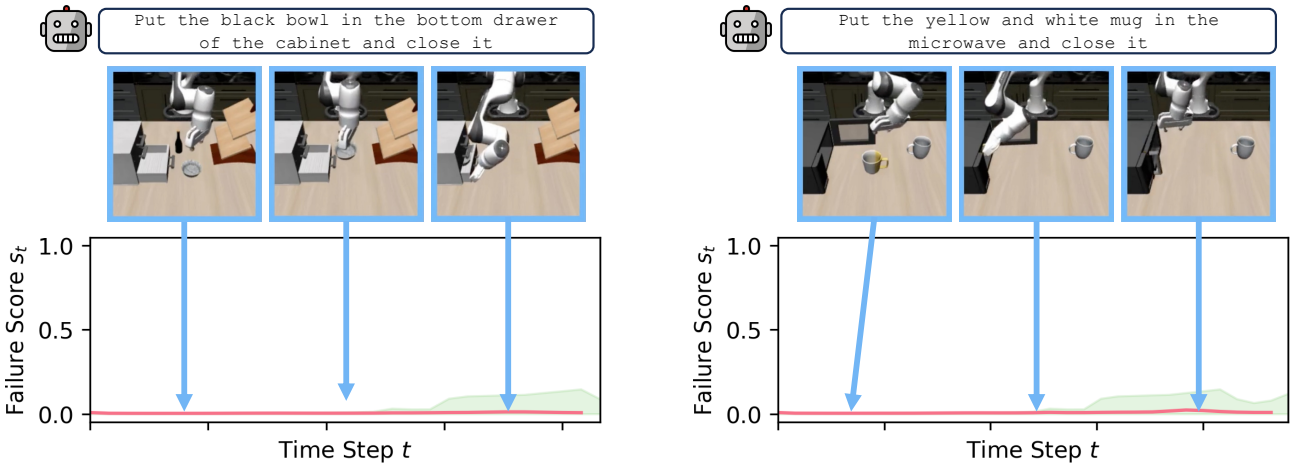
Figure 7. **Visualization of failure scores for the OpenVLA policy across different tasks.** A failure is declared at the earliest timestep t where the failure score s_t (red curve) exceeds the time-varying threshold ζ_t (green region) determined by conformal prediction.



(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.

C. Implementation Details

We implement Hide-and-Seek in Python 3.10.19 with PyTorch 2.10.0 (Paszke et al., 2019) on a single NVIDIA A6000 GPU with 48GB of memory, where training typically completes in approximately 5 minutes. We use the AdamW optimizer (Loshchilov & Hutter, 2019) 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 C.1.

C.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}

C.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 (Diquigiovanni et al., 2021). 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 (Diquigiovanni et al., 2021):

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

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 (Gu et al., 2025). For visualization clarity, we truncate the x-axis at the maximum detection-time endpoint reached by our method.

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

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 (Xu et al., 2025a; Gu et al., 2025). 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, 5 and 6).

C.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 (Römer et al., 2025) 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.

D. Additional Ablation Studies

D.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 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.

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	$0.885^{\pm 0.041}$	$0.926^{\pm 0.054}$	$0.693^{\pm 0.021}$	$0.892^{\pm 0.011}$	$0.921^{\pm 0.044}$	$0.705^{\pm 0.043}$

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}.

D.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.

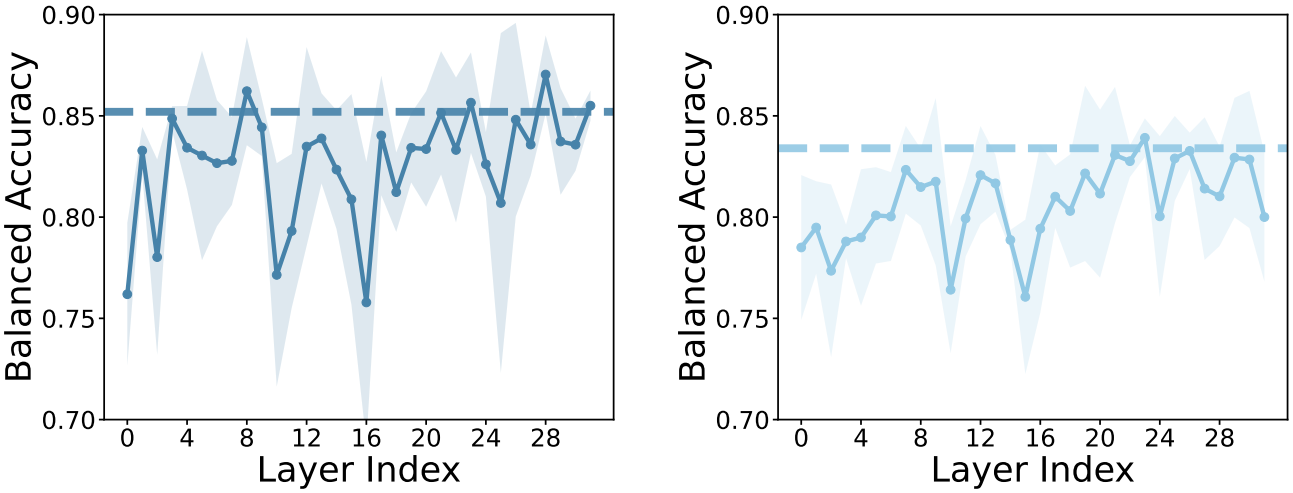
Policy: OpenVLA (Success Rate: 51.0%)						
	Seen Tasks			**Unseen Tasks**		
Thresholding	bACC↑	wACC↑	TWA↑	bACC↑	wACC↑	TWA↑
Fixed ($\zeta = 0.5$)	$0.859^{\pm 0.035}$	$0.864^{\pm 0.034}$	$0.656^{\pm 0.024}$	$0.792^{\pm 0.032}$	$0.780^{\pm 0.026}$	$0.625^{\pm 0.029}$
Split CP	$0.850^{\pm 0.030}$	$0.842^{\pm 0.028}$	$0.656^{\pm 0.011}$	$0.833^{\pm 0.026}$	$0.839^{\pm 0.022}$	$0.658^{\pm 0.005}$
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) (Angelopoulos & Bates, 2023), 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.

D.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.

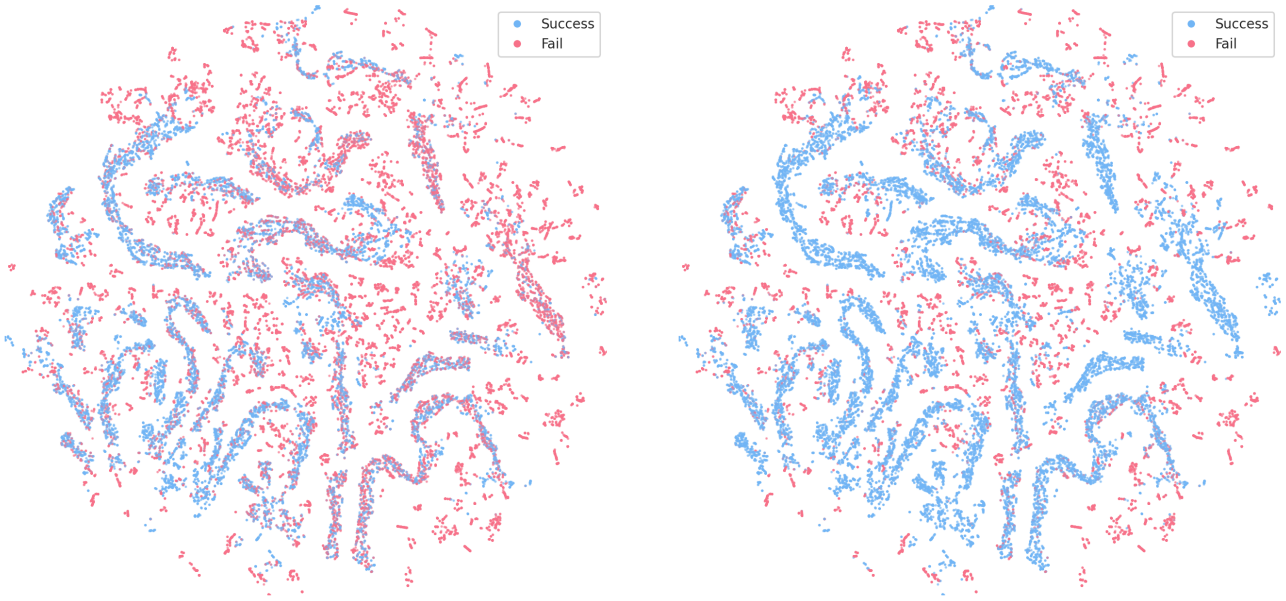
It is worth noting that recent work in LLM/LVLM interpretability has reported that intermediate layers often encode the most semantically rich features for downstream tasks (Skean et al., 2025; Jiang et al., 2025). While our results show a similar mild advantage for mid-to-late layers, a more principled investigation of which layers encode failure-discriminative information across different VLA architectures and policy types is an interesting direction for future work.

E. Additional Analysis

E.1. VLA Latent Visualization

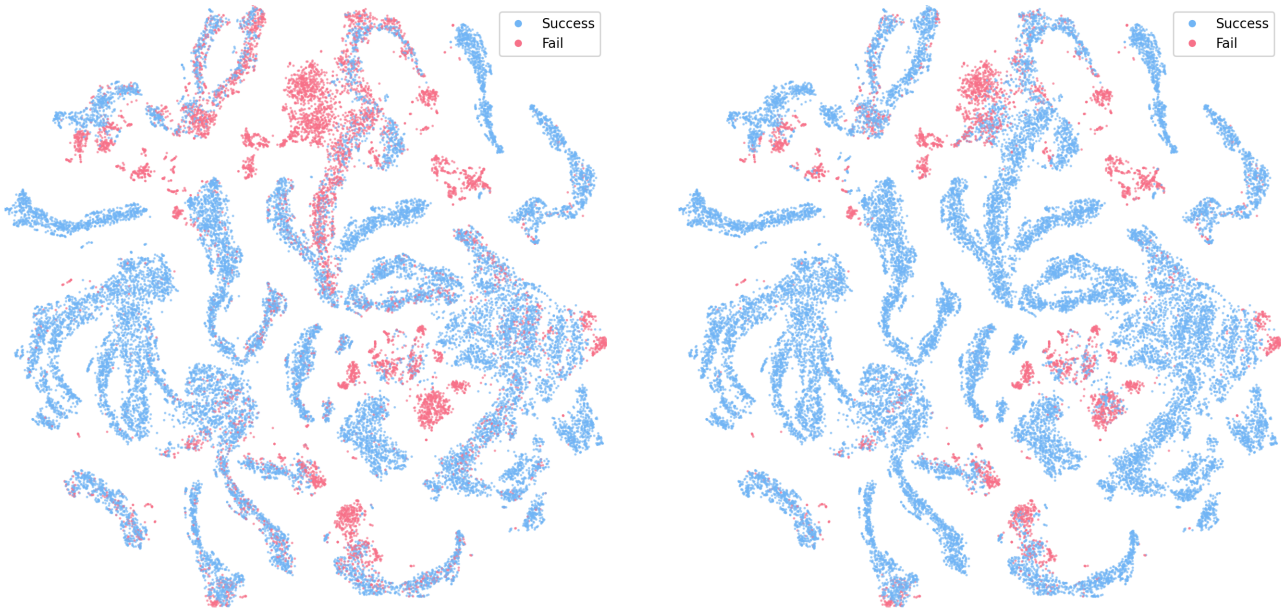
To better understand the impact of supervision granularity, we visualize the internal VLA action embeddings described in Section 5.1. We collect rollouts from all tasks in LIBERO-10 using OpenVLA and π_0, resulting in 500 episodes per policy (50 per task). At each timestep, we extract a compact representation $h_t \in \mathbb{R}^d$: 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 (Van der Maaten & Hinton, 2008) 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 G, 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



(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.**

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.

E.2. Detection Timeliness Relative to Annotated Failure Onset

Table 12. **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 G):

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

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 12, 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.

E.3. Detector Architecture

We provide detailed results and hyperparameter settings for the backbone architecture ablation (summarized in Table 2) in Table 13. We compare our LSTM-based design against three common alternatives: an MLP, a GRU (Chung et al., 2014), and a single-layer Transformer (Vaswani et al., 2017). 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 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.

F. Benchmark Details

Training and evaluation protocol. For both simulation (Section F.1) and real-robot (Section F.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

Table 13. 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	$\mathbf{0.885}^{\pm 0.041}$	$\mathbf{0.926}^{\pm 0.054}$	$\mathbf{0.693}^{\pm 0.021}$	$0.892^{\pm 0.011}$	$0.921^{\pm 0.044}$	$\mathbf{0.705}^{\pm 0.043}$

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 14.

Table 14. 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

F.1. Simulation Setup

We evaluate on two simulation benchmarks. **LIBERO** (Liu et al., 2023a) 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 (Kim et al., 2025) and π_0 (Black et al., 2024) using fine-tuned checkpoints released by the original authors. **VLABench** (Zhang et al., 2025) is a recent benchmark covering diverse manipulation tasks; we evaluate $\pi_{0.5}$ (Intelligence et al., 2025b) fine-tuned on its primary tasks. For both simulation benchmarks, we randomly hold out 3 out of 10 tasks as unseen for evaluation.

F.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 15, with sequential execution snapshots visualized in Table 16. We collect 25 demonstrations per task to fine-tune $\pi_{0.5}$ (Intelligence et al., 2025b) using LoRA (Hu et al., 2022). 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.

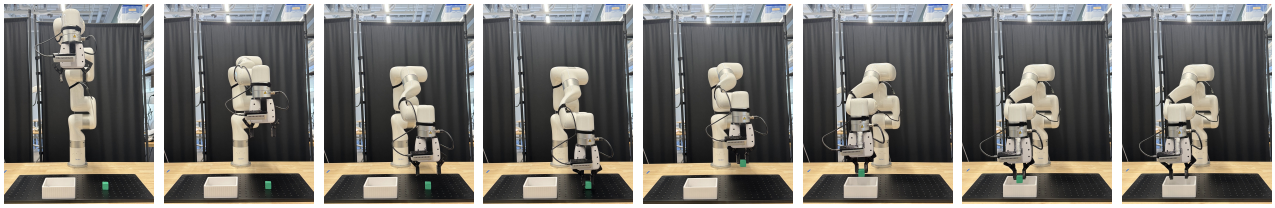
Table 15. List of tasks used in the real-world experiments.

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.

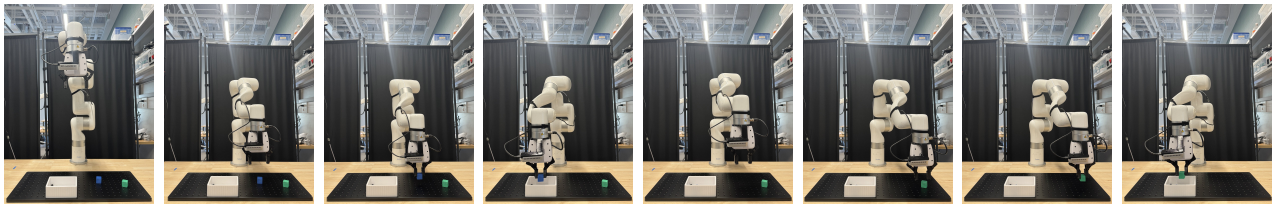
Table 16. Sequential execution snapshots for real-robot tasks. Each task is shown as 8 ordered frames (left-to-right). Tasks are grouped into two categories: CUBE (Tasks 1–4) and KITCHEN (Tasks 5–8).

CATEGORY 1: CUBE

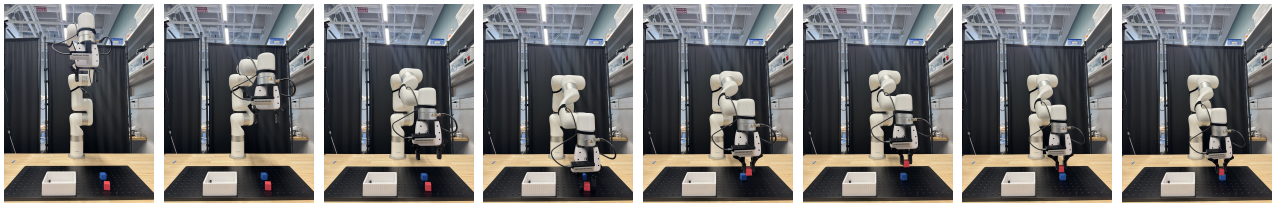
Task 1. Place the yellow cube into the container.



Task 2. Place the green cube into the container.

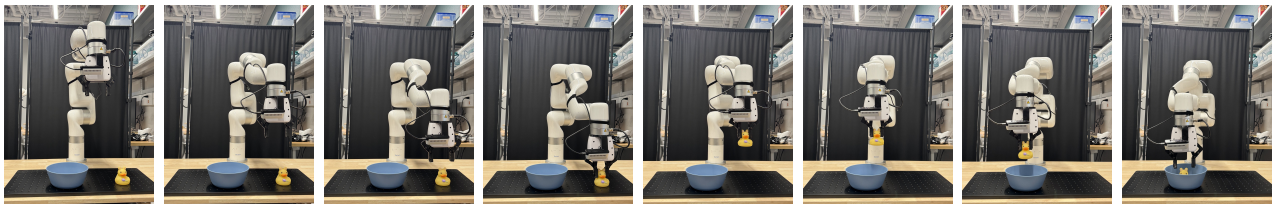


Task 3. Place both the blue cube and the green cube into the container.

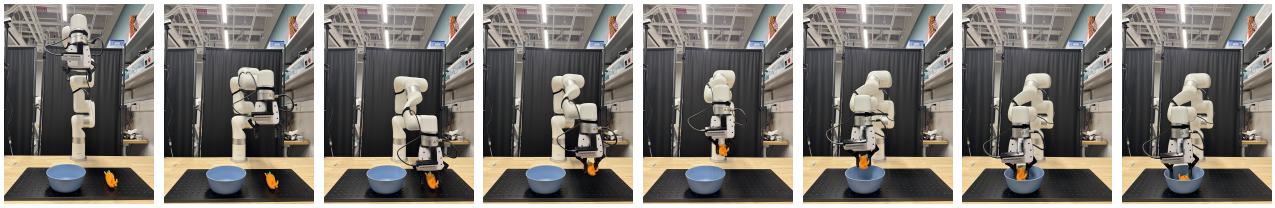


Task 4. Stack the red cube on top of the blue cube.

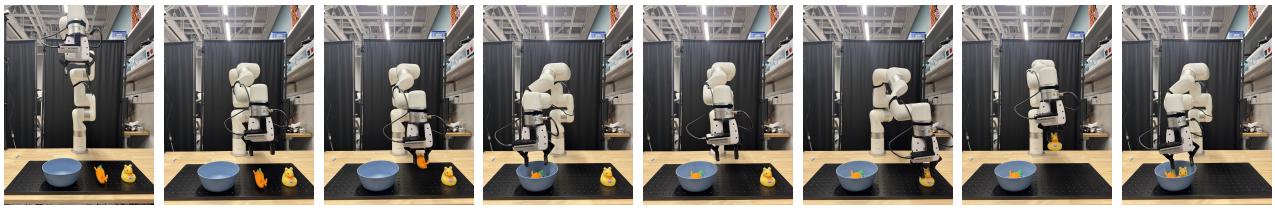
CATEGORY 2: KITCHEN



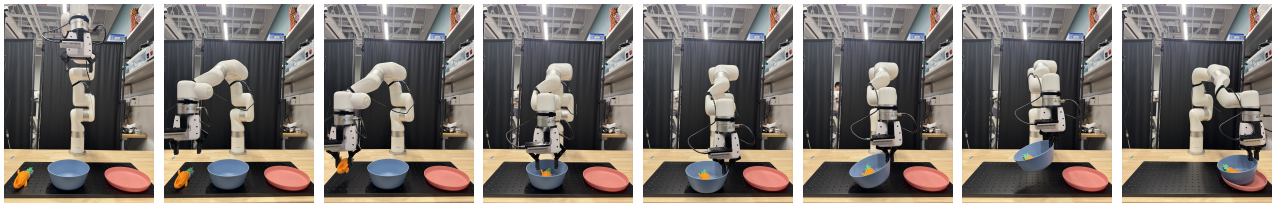
Task 5. Place the duck doll in the bowl.



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.

G. GPT-5.2-based Failure Onset Annotation

G.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 4, 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 12, 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 appears **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.

G.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 (Zheng et al., 2023), 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 12. 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.

H. 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 H.1) and classifier-based (Section H.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 H.2) operate on sampled actions $\{a_t^{(n)}\}_{n=1}^N$ drawn from the policy output; and token uncertainty-based methods (Section H.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.

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

H.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 (Gu et al., 2025). 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 (Lee et al., 2018). 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 (Sun et al., 2022). 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 (Liu et al., 2024c). 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.

H.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 (Gu et al., 2025) $f_{\text{succ}}^{\text{OOD}}$ 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) (He et al., 2024). 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 (Xu et al., 2025a). 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.

H.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 (Kuhn et al., 2023). Following the implementation of (Gu et al., 2025), 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 (Chen et al., 2024) 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) (Agia et al., 2024) 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) (Römer et al., 2025) 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.

H.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 (Gu et al., 2025) 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.

H.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 (Ren et al., 2023). 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 (Malinin & Gales, 2021). 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.

H.5. VLM-based Method

We implement a VLM-based runtime monitor using Qwen3-VL-8B-Instruct (Bai et al., 2025). 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).

I. 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 (Ji et al., 2023; Liu et al., 2024b). A broad line of work approaches this problem through uncertainty-based scoring functions. Logit-based methods use token-level probabilities as uncertainty estimates (Ren et al., 2023; Malinin & Gales, 2021; Zhou et al., 2024), verbalized methods prompt models to explicitly express their uncertainty in natural language (Lin et al., 2022; Groot & Valdenegro-Toro, 2024), and consistency-based methods measure agreement across multiple sampled responses (Kuhn et al., 2023; Chen et al., 2024; Zhang et al., 2024). 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 (Azaria & Mitchell, 2023; Du et al., 2024; Park et al., 2025; Niu et al., 2025).

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.

J. 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.

K. 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.