REAP: FROM VISUAL FAILURE RECOGNITION TO ACTION-CONDITIONED RISK PREDICTION FOR RUNTIME FAILURE MONITORING

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite rapid progress, generalist robot policies remain brittle during execution. Reliable deployment therefore requires runtime monitoring that provides timely warnings of failure. Existing monitors commonly estimate risk from current observations or policy signals without explicitly predicting the future consequences of proposed actions. To anticipate such risks before action execution, we introduce ReAP, a future-aware runtime monitoring framework that combines visual failure recognition with action-conditioned failure prediction. Specifically, ReAP first trains a visual detector on successful and failed rollouts to learn a failure-relevant representation space and estimate current risk. An action-conditioned predictor then forecasts the resulting future representation in this space to estimate future failure risk. At runtime, the detector and predictor provide complementary signals for current-state failure recognition and future-aware risk assessment. Experiments across diverse robot policies in simulation and the real world demonstrate that ReAP effectively balances failure-detection accuracy and warning timeliness.

1 INTRODUCTION

Recent advances in generalist embodied models and policies have substantially expanded the range of manipulation tasks that robots can perform across diverse environments and embodiments (Zitkovich et al., 2023; Kim et al., 2025; Black et al., 2024; NVIDIA et al., 2026). Despite this progress, learned policies remain vulnerable to unexpected visual changes, imperfect contacts, and compounding action errors, where seemingly small execution deviations can quickly lead to irreversible task failures (Agia et al., 2025). Runtime failure monitoring is therefore critical for reliable deployment. Timely warnings allow a robot to stop an unsafe action, replan, invoke a fallback strategy, or request human intervention before a recoverable deviation escalates into an irreversible task failure.

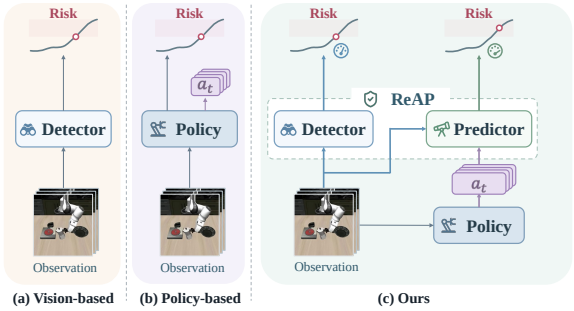


Figure 1: Vision-based monitors estimate failure risk from the current observation, while policy-based monitors rely on signals from the deployed policy. ReAP retains observation-based detection for assessing current risk and augments it with action-conditioned prediction to evaluate the future risk induced by a proposed action before execution.

Existing runtime monitors use different signals to estimate failure risk. Vision-based methods (Figure 1a) assess failure from the observed execution state (Duan et al., 2025; Zhou et al., 2026), while policy-based methods (Figure 1b) exploit signals such as policy-internal representations (Gu et al., 2025; Park et al., 2026) or uncertainty in the action-generation process (Lee et al., 2025; Rao et al., 2026). These approaches largely assess failure risk from evidence available at the current decision state. However, an execution may still appear nominal even when the proposed action would lead to failure, with the resulting deviation becoming visually apparent only after the action begins. Even from the same state, different actions may steer the execution toward diverse future outcomes,

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

such as recovery or further deviation. Thus, runtime monitoring calls for explicitly looking beyond current evidence and assessing the failure risk of the predicted future state before execution.

Motivated by this insight, we introduce **ReAP**, a future-aware runtime monitoring framework combining visual failure Recognition with Action-conditioned risk Prediction. The key idea is to estimate the future failure risk through action-conditioned prediction in a compact visual representation space learned for failure recognition. Specifically, we first train a visual detector on successful and failed rollouts to learn this representation that distinguishes successful and failed observations. Building on this representation, we follow the JEPA-style latent prediction paradigm (Assran et al., 2023) to train an action-conditioned predictor. Given the current visual representation and a proposed action chunk, the predictor models its consequences by estimating the detector representation of the resulting future observation. The frozen detector head is then reused to map this prediction to a future failure-risk score. As illustrated in Figure 1c, the detector assesses the observed state, whereas the predictor evaluates the proposed action before its outcome becomes observable.

We evaluate ReAP across four simulated policy–benchmark settings, covering OpenVLA (Kim et al., 2025) and π_0 (Black et al., 2024) on LIBERO-10 (Liu et al., 2023), and $\pi_{0.5}$ (Physical Intelligence et al., 2025) and Fast-WAM (Yuan et al., 2026) on eight selected RoboTwin tasks (Chen et al., 2025). We further deploy the monitor on a real robot across three manipulation tasks using $\pi_{0.5}$. Across all four simulation settings, our combined monitor consistently outperforms existing baselines, with relative improvements of up to $\mathbf{12.7}\%$ over the strongest evaluated baseline. Real-robot experiments further demonstrate the effectiveness of our ReAP in runtime monitoring with low inference overhead. To summarize, our main contributions are as follows:

1. We highlight that many runtime monitors rely only on current execution signals, limiting their ability to predict failures arising from the future consequences of proposed actions.
2. We introduce ReAP, a future-aware runtime monitoring framework that learns a failure-relevant visual representation and uses action-conditioned latent prediction to estimate the future failure risk associated with proposed actions (Section 4).
3. We demonstrate strong monitoring performance across diverse simulated policy–benchmark settings (Section 5.2) and real-world manipulation tasks (Section 5.3), together with detailed analyses of the detector and predictor components (Section 5.4).

2 Related Work

Runtime Failure Monitoring Existing runtime failure monitors estimate risk from different execution-time signals. Based on how these signals are processed to assess failure risk, existing approaches can be broadly grouped into four families. OOD-based methods detect deviations from successful or in-distribution behavior (Burda et al., 2019; Wong et al., 2022; Xu et al., 2025; Zhou et al., 2026). Action-uncertainty methods use signals from generated actions or the action-generation process, including action resampling, diffusion objectives, and flow-matching dynamics (Lee et al., 2025; Rao et al., 2026). Supervised detectors learn failure scores from labeled executions using visual, temporal, or policy representations (Farid et al., 2022; Gu et al., 2025; Park et al., 2026). Foundation-model-based approaches use pretrained language or vision-language models for semantic failure detection and reasoning (Sinha et al., 2024; Duan et al., 2025; Pacaud et al., 2025; Lin et al., 2025). Hybrid methods combine multiple signals. FIPER (Römer et al., 2025) combines observation novelty with action uncertainty, while SENTINEL (Agia et al., 2025) combines action uncertainty with VLM-based progress monitoring.

A separate line of predictive monitoring models future states under proposed actions. Liu et al. (2024) simulate future policy rollouts with latent dynamics. Concurrent works include Foresight (Zhang et al., 2026), which uses action-conditioned world-model features for failure detection, and ContactGuard (Zheng et al., 2026), which scores predicted post-contact latents with a failure probe. In comparison, ReAP first learns a failure-relevant representation space and models action consequences in this space to estimate future failure risk, considering both current visual signals and future action risks. Further comparisons are provided in Appendix A.

Test-Time Action Evaluation. A growing body of work evaluates candidate actions proposed by robot policies before execution. Recent approaches use learned verifiers, reward models, history-

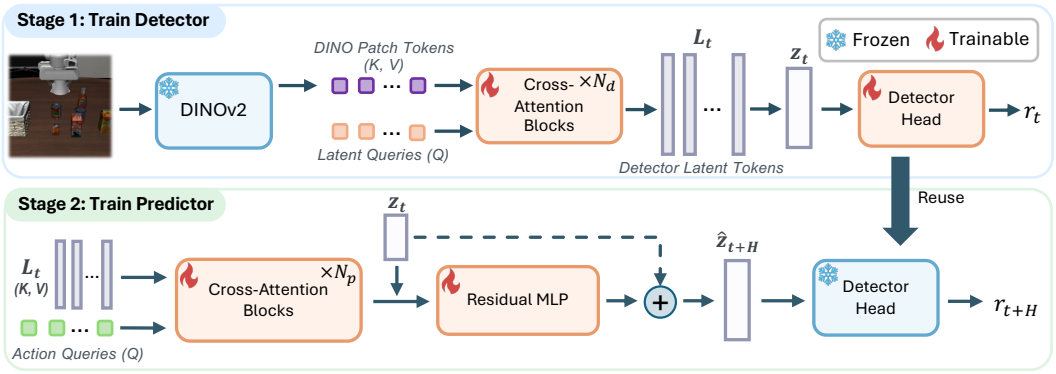


Figure 2: Overview of ReAP. Stage 1 learns an observation-based failure representation and detector. Stage 2 freezes the detector and learns an action-conditioned transition model that predicts future representations, which are evaluated by the same detector head.

aware verification, or model-internal confidence signals to score and select among multiple action proposals (Kwok et al., 2025; Dai et al., 2025; Li et al., 2025; Yang et al., 2025; Jang et al., 2026). These methods score candidate actions without explicitly predicting their future consequences. A complementary line of work predicts the consequences of candidate actions before evaluation. FOREWARN (Wu et al., 2025) uses a latent world model to imagine candidate outcomes and aligns the predicted latent states with a VLM for consequence evaluation. DynaGuide (Du & Song, 2025) predicts future latent states and uses their similarity to desired or undesired outcomes to guide action generation, while Do What You Say (Wu et al., 2026) forward-simulates candidate action sequences and evaluates whether the predicted outcomes agree with the policy's textual reasoning. Our method is most closely related to this predictive line, as we use this predictive capability as a runtime monitoring signal by evaluating the estimated representation in the failure-relevant space learned by the visual detector to assess risk before action execution.

3 Problem Setup

We consider runtime failure monitoring for robotic manipulation policies. When queried at environment timestep t, the deployed policy receives the visual observation O_t and proposes an action chunk A_t^π of length H_π. The observation-based detector estimates current-state risk r_t^{det}, while the action-conditioned predictor estimates future risk $r_{t+H|t}^{\text{pred}}$ associated with executing the first H actions of A_t^π, denoted by $A_t = (a_{t|t}, \ldots, a_{t+H-1|t})$, where H is the prediction horizon. Together, they support failure recognition from observed states and anticipation of risky action consequences before execution. The monitoring objective is to detect failed executions as early as possible while limiting false alarms on successful rollouts.

Both models are trained on successful and failed rollouts collected from the policy, using only episode-level outcomes. For rollout i of length T_i, the risk target is $y_{i,t} = 0$ if the rollout succeeds and $y_{i,t} = t/(T_i - 1)$ if it fails, where $t \in \{0, \ldots, T_i - 1\}$. This weak supervision avoids failure-onset annotation and assigns lower risk to early frames that may still appear nominal, with the target increasing to one at the end of a failed rollout.

4 Methods

As shown in Figure 2, our framework is trained in two stages. In Stage 1, an observation-only failure detector learns a failure-relevant visual representation using episode-level outcome supervision and maps it to a scalar risk score. In Stage 2, an action-conditioned transition model is trained in the latent space learned in Stage 1 to predict how the detector representation changes after executing a candidate action chunk. The predicted future representation is evaluated by the same frozen detector head, keeping the risk semantics of observed and predicted representations fixed. After training, thresholds for the detector and predictor are calibrated separately on held-out successful rollouts.

4.1 Observation-Only Failure Representation Learning

Stage 1 learns a policy-external representation of failure from the current visual observation O_t, without accessing candidate actions or internal policy features. Each observation may contain one or more synchronized camera views. For each camera view, dense patch tokens are extracted using a frozen DINOv2 (Oquab et al., 2023) backbone. The tokens are projected into a shared feature space and augmented with camera-specific and spatial positional embeddings. Tokens from all views are then concatenated to form the visual tokens of O_t.

To compress the dense visual features into a failure-relevant representation, we use a Perceiver-style feature extractor (Jaegle et al., 2021) with a fixed set of M learnable latent queries. The latent queries aggregate visual information through cross-attention, using the projected DINOv2 patch tokens as keys and values. This produces a fixed set of M detector latent tokens, denoted by L_t. We use $M = 32$ in all experiments. The detector latent tokens are first mean-pooled into a single vector that summarizes the current observation. A learned linear projection then maps this vector to the compact state representation z_t. Finally, the detector head maps z_t to the observation-level risk score $r_t^{\text{det}} \in [0, 1]$.

During Stage 1 training, the DINOv2 backbone remains frozen, while the visual projection, Perceiver-style feature extractor, and detector head are optimized using failure supervision. To train the detector, we use a class-weighted binary cross-entropy loss over valid observation timesteps:

$$\mathcal{L}_{\text{det}} = -\sum_i \sum_t \left[w_+ y_{i,t} \log r_{i,t}^{\text{det}} + w_-(1 - y_{i,t}) \log(1 - r_{i,t}^{\text{det}}) \right], \tag{1}$$

where $y_{i,t}$ is the risk target defined in Section 3, and w_+ and w_- compensate for class imbalance.

4.2 Action-Conditioned Latent Dynamics in Failure Space

Stage 2 models how a candidate action chunk changes the failure-relevant representation learned in Stage 1. Inspired by JEPA-style latent prediction (Assran et al., 2023; Bardes et al., 2024), we predict the representation of the future observation directly in the learned latent space rather than reconstructing the observation itself. The complete Stage 1 detector is frozen throughout this stage.

Given the current compact representation z_t and the action proposal A_t available from the current policy query, each action is independently projected to the transition latent dimension and augmented with a learnable step embedding that encodes its position within the action chunk. The resulting action tokens serve as queries in stacked cross-attention blocks. To construct the keys and values, we apply linear projections independently to each token in L_t. The cross-attention blocks attend to these projected tokens and produce action-conditioned features over the prediction horizon. We mean-pool both the action tokens and the resulting action-conditioned features, and concatenate them with the current compact representation z_t. A residual transition MLP takes the concatenated features as input and predicts a latent update, which is added to the current representation z_t to obtain the future compact representation $\hat{z}_{t+H|t}$. The predicted representation is passed through the frozen Stage 1 detector head to obtain the anticipated future risk $r_{t+H|t}^{\text{pred}}$.

During training, the actual future observation O_{t+H} is processed by the same frozen detector to obtain the target representation z_{t+H}. The transition model is trained to predict this future latent representation using an MSE objective together with future-risk supervision:

$$\mathcal{L}_{\text{pre}} = \frac{\lambda_{\text{feat}}}{d} \left\| \hat{z}_{t+H|t} - \text{sg}(z_{t+H}) \right\|_2^2 + \lambda_{\text{risk}} \mathcal{L}_{\text{det}}^{\text{future}}. \tag{2}$$

Here, d denotes the compact state dimension and sg stops gradients through the target representation. The future-risk loss $\mathcal{L}_{\text{det}}^{\text{future}}$ follows Equation 1 using $r_{t+H|t}^{\text{pred}}$. The two terms supervise future latent prediction and failure-risk estimation, respectively. We use $\lambda_{\text{feat}} = \lambda_{\text{risk}} = 1$ by default.

4.3 Threshold Calibration

Following prior work on time-dependent failure-score calibration (Xu et al., 2025), we calibrate monitoring thresholds using a held-out set of successful rollouts. Scores estimated may vary across

execution phases even in successful rollouts, motivating thresholds that vary with time. To align score trajectories for calibration, we consider three temporal parameterizations. *Absolute-time* calibration indexes thresholds by the raw timestep t. *Normalized-time* calibration represents each timestep by its relative progress within the trajectory. *Staged-time* calibration partitions normalized execution progress into a fixed number of stages and uses a shared threshold within each stage.

For each temporal parameterization, we construct thresholds using either *pointwise* quantiles or a one-sided *functional* conformal band, resulting in six calibration variants. The detector and predictor are calibrated independently under the same protocol. For further details, refer to Appendix E. Let δ_t^{det} and δ_t^{pred} denote their respective thresholds. At runtime, the monitor raises an alarm whenever either branch crosses its calibrated threshold:

$$b_t = 1\left[r_t^{\mathrm{det}} \geq \delta_t^{\mathrm{det}}\right] \vee 1\left[r_{t+H|t}^{\mathrm{pred}} \geq \delta_t^{\mathrm{pred}}\right]. \tag{3}$$

5 Experiments

We evaluate ReAP in simulation and on real robots to answer three questions. (1) Does ReAP achieve a favorable balance between failure-detection accuracy, warning timeliness, and false alarms across diverse policies and tasks? (Section 5.2 and Appendix D.1) (2) Can ReAP maintain effective online monitoring on real robots while introducing low inference overhead? (Section 5.3) (3) What enables its monitoring performance, and how do failure-relevant representation learning and action-conditioned prediction contribute to the final monitor? (Section 5.4 and Appendix D.2)

5.1 Experiment Setup

Baselines. We compare ReAP with representative runtime failure monitors from three categories. *Action-uncertainty methods* estimate failure risk from the policy's action-generation process. STAC (Agia et al., 2025) measures inconsistency across sampled actions, Diff-DAgger (Lee et al., 2025) uses diffusion-model objectives as uncertainty signals, and ACCEL (Rao et al., 2026) derives uncertainty from flow-matching trajectories. *OOD-based methods* identify failures from deviations relative to successful execution. We include RND (Burda et al., 2019) and LogpZO (Xu et al., 2025), and additionally evaluate FIPER (Römer et al., 2025), which augments an observation-based OOD signal with action uncertainty. *Classifier-based methods* directly learn failure risk from successful and failed rollouts. We include SAFE-MLP and SAFE-LSTM (Gu et al., 2025), which operate on internal VLA representations extracted from the deployed policy.

Evaluation Protocols. Following Römer et al. (2025), we evaluate runtime monitoring with three complementary metrics, balanced time-weighted accuracy (bTWA), balanced accuracy (bACC), and normalized detection time (T-det). bACC measures failure-detection accuracy while accounting for false alarms on successful rollouts, with $\mathrm{bACC} = (\mathrm{TPR} + \mathrm{TNR})/2$. T-det measures warning timeliness as the mean normalized first-alarm time over all failed rollouts, with missed failures assigned a value of one. We use bTWA as the primary metric because it jointly rewards failure detection and early warning while penalizing missed failures and false alarms:

$$\mathrm{bTWA} = \frac{1}{2}\left(\frac{1}{N_{\mathrm{fail}}}\sum_{i\in\mathrm{TP}}\left(1-\frac{t_i}{T_i}\right) + \frac{N_{\mathrm{TN}}}{N_{\mathrm{succ}}}\right), \tag{4}$$

Here, TP denotes the set of failed rollouts that are detected, and t_i and T_i denote the first-alarm time and rollout length for a correctly detected failure. N_{fail}, N_{succ} and N_{TN} denote the numbers of failed rollouts, successful rollouts, and successful rollouts without false alarms, respectively.

5.2 Offline Failure Monitoring

We evaluate offline failure monitoring across four policy–benchmark settings. On LIBERO-10 (Liu et al., 2023), we evaluate OpenVLA (Kim et al., 2025) and π_0 (Black et al., 2024). On RoboTwin 2.0 (Chen et al., 2025), we evaluate $\pi_{0.5}$ (Physical Intelligence et al., 2025) and Fast-WAM (Yuan et al., 2026) on policy-specific subsets of eight tasks, referred to as RoboTwin-8. For each setting, a number of successful rollouts are held out for threshold calibration, and all reported results are

Table 1: Offline failure monitoring across four policy–benchmark settings. Results are averaged over five calibration levels, $\alpha \in \{0.05, 0.10, 0.15, 0.20, 0.25\}$, with means and standard deviations reported over three seeds. Best and second-best results for each metric are shown in **bold** and underline, respectively. ReAP (OR) is the complete monitor, while the detector and predictor rows report standalone performance for reference. A dash (–) indicates that the method is not applicable to the corresponding policy in our evaluation setting, with further details provided in Appendix C. RoboTwin-8 denotes policy-specific eight-task subsets selected separately for $\pi_{0.5}$ and Fast-WAM.

(a) LIBERO-10

		OpenVLA (Success Rate: 53.6%)			π_0 (Success Rate: 80.4%)		
Category	Method	bTWA ↑	bACC ↑	T-det ↓	bTWA ↑	bACC ↑	T-det ↓
Action Uncertainty	STAC	–	–	–	$0.490^{\pm 0.000}$	$0.578^{\pm 0.000}$	$\underline{0.235}^{\pm 0.000}$
	Diff-DAgger	–	–	–	$0.454^{\pm 0.000}$	$0.531^{\pm 0.000}$	$\mathbf{0.163}^{\pm 0.000}$
	ACCEL	–	–	–	$0.628^{\pm 0.000}$	$0.777^{\pm 0.000}$	$0.524^{\pm 0.000}$
OOD-based	RND	$0.584^{\pm 0.013}$	$0.715^{\pm 0.048}$	$\underline{0.596}^{\pm 0.069}$	$0.751^{\pm 0.013}$	$\mathbf{0.874}^{\pm 0.010}$	$0.410^{\pm 0.027}$
	LogpZO	$0.475^{\pm 0.023}$	$0.531^{\pm 0.019}$	$0.809^{\pm 0.114}$	$0.714^{\pm 0.006}$	$0.862^{\pm 0.007}$	$0.454^{\pm 0.008}$
	FIPER	$0.559^{\pm 0.004}$	$\mathbf{0.808}^{\pm 0.035}$	$0.812^{\pm 0.015}$	$0.683^{\pm 0.005}$	$0.866^{\pm 0.005}$	$0.518^{\pm 0.022}$
Classifier	SAFE-MLP	$\underline{0.622}^{\pm 0.004}$	$0.687^{\pm 0.001}$	$0.641^{\pm 0.019}$	$\underline{0.758}^{\pm 0.005}$	$0.869^{\pm 0.005}$	$0.378^{\pm 0.005}$
	SAFE-LSTM	$0.502^{\pm 0.004}$	$0.528^{\pm 0.037}$	$0.889^{\pm 0.077}$	$0.724^{\pm 0.014}$	$\underline{0.871}^{\pm 0.010}$	$0.394^{\pm 0.024}$
Ours	**ReAP (Detector)**	$0.671^{\pm 0.021}$	$0.744^{\pm 0.020}$	$0.450^{\pm 0.041}$	$0.750^{\pm 0.005}$	$0.838^{\pm 0.013}$	$0.466^{\pm 0.014}$
	ReAP (Predictor)	$0.683^{\pm 0.007}$	$0.764^{\pm 0.017}$	$0.406^{\pm 0.038}$	$0.755^{\pm 0.009}$	$0.831^{\pm 0.010}$	$0.461^{\pm 0.017}$
	ReAP (OR)	$\mathbf{0.701}^{\pm 0.010}$	$\underline{0.770}^{\pm 0.008}$	$\mathbf{0.326}^{\pm 0.029}$	$\mathbf{0.780}^{\pm 0.008}$	$0.865^{\pm 0.015}$	$0.395^{\pm 0.025}$

(b) RoboTwin-8

		$\pi_{0.5}$ (Success Rate: 48.4%)			Fast-WAM (Success Rate: 66.6%)		
Category	Method	bTWA ↑	bACC ↑	T-det ↓	bTWA ↑	bACC ↑	T-det ↓
Action Uncertainty	STAC	$0.507^{\pm 0.000}$	$0.554^{\pm 0.000}$	$0.349^{\pm 0.000}$	$0.487^{\pm 0.000}$	$0.506^{\pm 0.000}$	$\mathbf{0.269}^{\pm 0.000}$
	Diff-DAgger	$0.513^{\pm 0.000}$	$0.560^{\pm 0.000}$	$\underline{0.312}^{\pm 0.000}$	$0.438^{\pm 0.044}$	$0.460^{\pm 0.034}$	$0.921^{\pm 0.012}$
	ACCEL	$0.512^{\pm 0.000}$	$0.644^{\pm 0.000}$	$0.675^{\pm 0.000}$	$0.491^{\pm 0.000}$	$0.556^{\pm 0.000}$	$0.883^{\pm 0.000}$
OOD-based	RND	$\underline{0.667}^{\pm 0.000}$	$0.733^{\pm 0.007}$	$0.636^{\pm 0.022}$	$0.537^{\pm 0.038}$	$0.585^{\pm 0.053}$	$0.605^{\pm 0.108}$
	LogpZO	$0.645^{\pm 0.009}$	$0.775^{\pm 0.009}$	$0.556^{\pm 0.004}$	$0.545^{\pm 0.010}$	$0.561^{\pm 0.010}$	$0.771^{\pm 0.036}$
	FIPER	$0.645^{\pm 0.024}$	$\mathbf{0.815}^{\pm 0.035}$	$0.493^{\pm 0.065}$	$0.541^{\pm 0.005}$	$0.597^{\pm 0.012}$	$\underline{0.590}^{\pm 0.039}$
Classifier	SAFE-MLP	$0.589^{\pm 0.004}$	$0.668^{\pm 0.006}$	$0.605^{\pm 0.004}$	$0.551^{\pm 0.014}$	$0.559^{\pm 0.017}$	$0.719^{\pm 0.035}$
	SAFE-LSTM	$0.562^{\pm 0.010}$	$0.598^{\pm 0.012}$	$0.701^{\pm 0.003}$	$\underline{0.578}^{\pm 0.009}$	$\underline{0.636}^{\pm 0.012}$	$0.800^{\pm 0.007}$
Ours	**ReAP (Detector)**	$0.689^{\pm 0.021}$	$0.795^{\pm 0.018}$	$0.356^{\pm 0.027}$	$0.626^{\pm 0.013}$	$0.752^{\pm 0.015}$	$0.675^{\pm 0.038}$
	ReAP (Predictor)	$0.695^{\pm 0.017}$	$0.792^{\pm 0.019}$	$0.342^{\pm 0.027}$	$0.640^{\pm 0.015}$	$0.761^{\pm 0.008}$	$0.642^{\pm 0.053}$
	ReAP (OR)	$\mathbf{0.701}^{\pm 0.013}$	$\underline{0.785}^{\pm 0.011}$	$\mathbf{0.306}^{\pm 0.021}$	$\mathbf{0.651}^{\pm 0.014}$	$\mathbf{0.779}^{\pm 0.008}$	$0.592^{\pm 0.051}$

computed on a separate evaluation split. Following Park et al. (2026), we average performance over $\alpha \in \{0.05, 0.10, 0.15, 0.20, 0.25\}$ under the same calibration protocol unless otherwise specified.

Results Analysis. Table 1 shows that ReAP achieves the highest bTWA across all four policy–benchmark settings, with relative improvements of up to 12.7% over the strongest evaluated baseline in each setting. Averaged across the four settings, ReAP achieves a bTWA of 0.708, a bACC of 0.800, and a T-det of 0.405. These results demonstrate a favorable balance between failure-detection accuracy and warning timeliness across different policy architectures and manipulation benchmarks.

The baseline results reveal different limitations in achieving this balance between accuracy and timeliness. Action-uncertainty methods can produce early warnings but often with lower balanced accuracy. For example, Diff-DAgger reaches a T-det of 0.163 on π_0/LIBERO-10, while its bACC is only 0.531. In contrast, OOD-based methods often achieve stronger balanced accuracy but raise alarms later. FIPER obtains a bACC of 0.808 on OpenVLA/LIBERO-10, but with a T-det of 0.812. SAFE is competitive in several settings, yet its performance varies across policies.

To assess how the two branches contribute to these gains, we also report the results of the detector and predictor. Both branches of ReAP already provide strong standalone monitoring signals. The detector effectively recognizes failure evidence from observed execution states, while the predictor achieves comparable or stronger performance through action-conditioned consequence prediction.

Table 2: Real-world failure monitoring and runtime efficiency. (a) Failure monitoring results across three real-world manipulation tasks. SR denotes the success rate of the deployed base policy. (b) Mean runtime per monitoring evaluation for representative failure monitors on a single NVIDIA A800 80GB GPU. For methods that rely on policy-derived features or action-uncertainty signals, the reported runtime includes the policy inference required to obtain these signals.

(a) Real-world Failure Monitoring

	Fold Cloth (SR 60.0%)		**Stack Plates (SR 75.0%)**		**Pick Cubes (SR 50.0%)**	
Method	bTWA ↑	bACC ↑	bTWA ↑	bACC ↑	bTWA ↑	bACC ↑
STAC	0.542	0.613	0.599	0.721	0.518	0.546
ACCEL	0.584	0.639	0.407	0.517	0.452	0.525
RND	0.566	0.664	0.597	0.721	0.608	0.695
LogpZO	0.541	0.650	0.754	0.893	0.547	0.632
FIPER	0.508	0.513	0.671	0.759	0.668	**0.800**
SAFE-MLP	**0.624**	0.688	0.801	0.892	0.572	0.681
ReAP (OR)	0.603	**0.713**	**0.818**	**0.900**	**0.718**	**0.800**

(b) Runtime Comparison

Method	Runtime (ms) ↓
Base policy ($\pi_{0.5}$)	67.07
STAC	223.92
Diff-DAgger	775.51
ACCEL	87.98
RND	82.80
LogpZO	76.68
FIPER	224.69
SAFE-MLP	67.00
ReAP (Detector)	**9.72**
ReAP (Predictor)	81.16

Applying the OR rule further improves bTWA over both components in all four settings and consistently reduces normalized detection time. These consistent gains support the complementary roles of current-state failure recognition and action-conditioned future-risk prediction.

5.3 Online Real-Robot Monitoring

We further evaluate ReAP on three real-world manipulation tasks, *Fold Cloth*, *Stack Plates*, and *Pick Cubes*, using separately fine-tuned $\pi_{0.5}$ policies. For each task, we collect successful and failed policy rollouts to train and calibrate the monitor, and deploy ReAP during physical robot execution. During physical robot execution, the detector continuously monitors incoming observations, while the predictor evaluates each proposed action chunk before execution. Monitoring alarms are recorded without altering robot execution. To enable controlled comparison on identical physical trajectories, all baselines are evaluated post hoc using the same recorded rollouts and policy outputs. Additional details of the real-robot setup and deployment are provided in Appendix C.4.

Table 2a shows that ReAP maintains strong monitoring performance across all three real-world tasks. It achieves the highest bTWA on Stack Plates and Pick Cubes, and obtains the best or tied-best bACC on all three tasks. Averaged across the three tasks, ReAP achieves a bTWA of 0.713 and a bACC of 0.804, supporting its effectiveness in real-world failure monitoring.

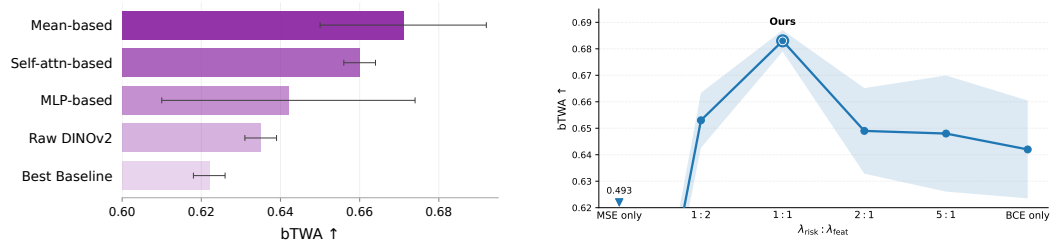
We also benchmark monitoring latency using $\pi_{0.5}$ as the reference policy. As shown in Table 2b, the observation-only detector runs in 9.72 ms without requiring policy inference. Predictor-based monitoring takes 81.16 ms per policy query, including the 67.07 ms base-policy inference, corresponding to an additional 14.09 ms in this benchmark. In comparison, methods based on repeated policy sampling require substantially more computation. These measurements show that ReAP can provide effective online monitoring while adding limited latency to the deployed control pipeline.

5.4 Component Analysis

This section examines the main factors underlying the effectiveness of ReAP. The detector analysis studies whether failure supervision produces a representation that is more suitable for failure monitoring than generic visual or policy features. The predictor analysis then evaluates how this representation benefits future-state prediction and how action conditioning improves monitoring performance. All component analyses are conducted on OpenVLA/LIBERO-10 under the same evaluation protocol as the offline benchmark. Additional analyses are provided in Appendix D.2

5.4.1 Detector Representation Learning

Architecture Ablation. We ablate the feature extractor and latent-token aggregation in the observation-only detector. The *Raw DINOv2* baseline directly mean-pools frozen DINOv2 patch features without the Perceiver-style feature extractor. The remaining variants use the Perceiver-style



(a) (b)

Figure 3: (a) Effect of different detector architectures. (b) Effect of future-risk and latent-transition loss weighting on predictor performance. The downward triangle indicates an off-scale value.

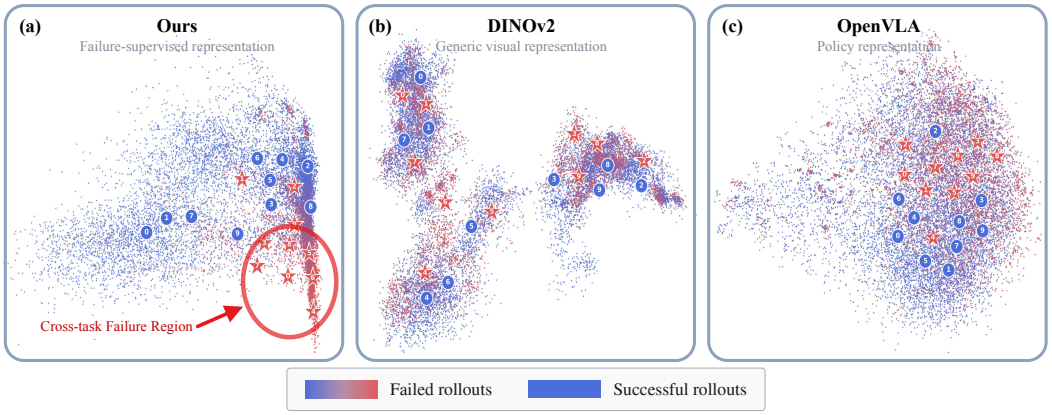


Figure 4: PCA visualization of representations on OpenVLA/LIBERO-10. (a) Detector features. (b) DINOv2 features. (c) OpenVLA features. Observations from successful rollouts are shown in blue, while those from failed rollouts are colored from blue to red as execution progresses. Blue circles denote task-wise mean representations over observations from successful rollouts. Red stars denote task-wise mean representations over the final 30% of frames from failed rollouts.

extractor to produce a fixed set of latent tokens and differ only in how these tokens are aggregated, using an MLP, self-attention, or mean pooling. As shown in Figure 3a, directly mean-pooling frozen DINOv2 features already achieves a bTWA of 0.635, exceeding the strongest baseline in the main benchmark at 0.622. Introducing the failure-supervised feature extractor further improves bTWA to 0.671 with mean pooling, supporting the benefit of learning monitoring-relevant features on top of the frozen visual backbone. Mean pooling performs best among the aggregation variants, and we adopt it together with the learned feature extractor in the default detector.

Representation Structure. To examine how failure supervision separates successful and failed observations, we compare the learned detector representation with frozen DINOv2 patch features and OpenVLA internal representations in Figure 4. In the 2D PCA projection, DINOv2 features are organized primarily by task appearance, with successful and failed observations from the same task remaining close to each other. OpenVLA features similarly show limited separation by execution outcome. In contrast, the learned detector representation forms a more coherent failure region shared across tasks, with late-stage observations from failed rollouts clustering more closely in the projected space. Together with the quantitative gains in Figure 3a, this suggests that the learned representation captures failure-related structure shared across the evaluated tasks.

5.4.2 Action-Conditioned Predictor Learning

Training Objective. The predictor is trained with two complementary objectives. The latent-transition loss encourages the predicted representation to match the detector representation of the actual future observation, while the future-risk loss directly supervises the predicted failure risk. We

Table 3: (a) Effect of the detector representation. (b) Effect of action conditioning.

(a) Prediction Representation

Method	bTWA ↑	bACC ↑
Generic PCA	$0.644^{\pm 0.012}$	$0.747^{\pm 0.028}$
Random detector, $\lambda_{\text{feat}} = 0$	$0.641^{\pm 0.012}$	$0.742^{\pm 0.027}$
Random detector	$0.650^{\pm 0.005}$	$0.752^{\pm 0.009}$
Learned detector (Ours)	$\mathbf{0.683}^{\pm 0.007}$	$\mathbf{0.764}^{\pm 0.017}$

(b) Action Conditioning

Variant	bTWA ↑	bACC ↑
Zero action, MSE+BCE	$0.520^{\pm 0.007}$	$0.603^{\pm 0.016}$
Zero action, BCE only	$0.541^{\pm 0.008}$	$0.632^{\pm 0.015}$
Action shuffle	$0.618^{\pm 0.005}$	$0.665^{\pm 0.006}$
Action-conditioned (Ours)	$\mathbf{0.683}^{\pm 0.007}$	$\mathbf{0.764}^{\pm 0.017}$

vary the relative weighting $\lambda_{\text{risk}} : \lambda_{\text{feat}}$ while keeping the predictor architecture fixed. As shown in Figure 3b, using latent-transition supervision alone yields a bTWA of 0.493, whereas future-risk supervision alone reaches 0.642. Combining the two objectives further improves bTWA to 0.683, with equal weighting performing best among the tested configurations. These results indicate that direct future-risk supervision is important for effective monitoring, and that jointly predicting future detector representations further improves performance over risk-only training.

Prediction Representation. Beyond the training objective, we assess how the representation space affects future-risk prediction by comparing three variants with the Stage 2 predictor architecture fixed. *Generic PCA* compresses frozen DINOv2 features using PCA, *Random detector* uses the same detector architecture without failure supervision, and *Learned detector* uses the representation obtained after Stage 1 training. For the first two variants, the corresponding detector head is learned jointly with the predictor. As shown in Table 3a, predicting in the learned detector space achieves the strongest performance, reaching a bTWA of 0.683 and a bACC of 0.764, compared with 0.650 and 0.752 for the random detector and 0.644 and 0.747 for Generic PCA. Adding the latent-transition loss to the random-detector variant yields only a small increase in mean bTWA. Together with the stronger performance of the learned detector, this suggests that failure-supervised detector features provide more effective prediction targets for future-risk estimation than generic or randomly initialized representations.

Action Conditioning. Finally, we examine the contribution of action conditioning through two complementary ablations. First, we retrain the predictor with the entire action chunk set to zero. As shown in Table 3b, retaining the MSE+BCE objective reduces bTWA from 0.683 to 0.520 and bACC from 0.764 to 0.603. Training the zero-action variant with BCE alone also leaves a substantial gap to the full action-conditioned predictor. Second, we keep the trained predictor fixed and replace the action chunk at evaluation time with one sampled from another rollout of the same task, which preserves task identity while disrupting the correspondence between the current observation and the proposed action. This action shuffle reduces bTWA to 0.618 and bACC to 0.665. It also increases both L_2 error and per-dimension MSE against the unchanged target future latent in all three seeds, as shown in Table 4. Together, these results support the role of action conditioning in predicting the observed future representation and improving failure monitoring.

Table 4: Future-latent prediction errors.

Action	L_2 error ↓	Per-dim. MSE ↓
Shuffled	$10.349^{\pm 0.038}$	$0.445^{\pm 0.003}$
Original	$\mathbf{9.680}^{\pm 0.011}$	$\mathbf{0.394}^{\pm 0.001}$

6 Conclusion

We presented ReAP, a future-aware runtime failure monitoring framework that builds on visual failure recognition to predict future action risks. A visual detector learns a failure-relevant representation space from successful and failed rollouts, within which an action-conditioned predictor estimates future representations and their associated risk. The combined monitor achieves the highest bTWA across all four simulated policy–benchmark settings, supporting the complementary value of current-state recognition and future-risk prediction in balancing failure detection, warning timeliness, and false alarms. Real-robot experiments demonstrate effective online monitoring, and runtime measurements indicate limited inference overhead. Component analyses further support the roles of failure-supervised representation learning and action conditioning in improving future-latent prediction and failure-risk estimation.

AI USE STATEMENT

In this work, we used generative AI tools to assist with manuscript drafting and language refinement, literature retrieval and discovery, discussions of research ideas and experimental design, and analysis and interpretation of experimental results. Generative AI tools were also used to support code editing and figure prototyping during the development of the paper. We also used Qwen3.5-9B to obtain approximate failure-onset annotations for the supplementary timing analysis (Appendix D.3). All research ideas, methodological decisions, experiments, quantitative results, and scientific claims were reviewed and verified by the authors. Literature suggestions were checked against the original sources, AI-assisted code was manually inspected, and AI-assisted interpretations of experimental results were verified against the underlying data. We did not use generative AI tools to generate synthetic datasets, formulate or prove mathematical claims, assist with mathematical proofs, or clean and reformat datasets. We take full responsibility for all AI-assisted content in this work.

REPRODUCIBILITY STATEMENT

The method formulation and training objectives are described in Sec. 3 and 4, and the experimental setup and evaluation protocol are provided in Sec. 5.1. Appendix E provides the detailed construction and theoretical discussion of our threshold calibration. Additional implementation details are provided in Appendix C, including benchmark construction, baseline implementations, model architecture and training configurations, and the real-robot deployment setup. Code is provided in the supplementary material.

REFERENCES

ACE-Brain Team, Ziyang Gong, Haoming Gu, Zehang Luo, Tianyi Zhang, Tao Tao, Yixiao Chi, Zhe Liu, Lingsi Zhu, Jingyuan Liu, Anke Tang, Songze Li, Yilun Kong, Ningjing Liu, Tianyu Zhu, Yunpeng Qing, Shuang Luo, Xiang Liu, Shi Fu, Dawei Nie, Sixiang Liu, Zhexi Wen, Feng Pan, Xiaofeng Wang, Zhi Hou, Chunxiao Liu, Xue Yang, Junchi Yan, Hengshuang Zhao, Dacheng Tao, and Xiaogang Wang. ACE-Brain-0.5: A unified embodied foundational model for physical agentic AI. *arXiv preprint arXiv:2607.04426*, 2026. URL https://arxiv.org/abs/2607.04426.

Christopher Agia, Rohan Sinha, Jingyun Yang, Ziang Cao, Rika Antonova, Marco Pavone, and Jeannette Bohg. Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 689–723. PMLR, 2025. URL https://proceedings.mlr.press/v270/agia25a.html.

Anastasios N. Angelopoulos and Stephen Bates. Conformal prediction: A gentle introduction. *Foundations and Trends in Machine Learning*, 16(4):494–591, 2023. doi: 10.1561/2200000101. URL https://doi.org/10.1561/2200000101.

Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15619–15629, 2023.

Mahmoud Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, Sergio Arnaud, Abha Gejji, Ada Martin, Francois Robert Hogan, Daniel Dugas, Piotr Bojanowski, Vasil Khalidov, Patrick Labatut, Francisco Massa, Marc Szafraniec, Kapil Krishnakumar, Yong Li, Xiaodong Ma, Sarath Chandar, Franziska Meier, Yann LeCun, Michael Rabbat, and Nicolas Ballas. V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025. URL https://arxiv.org/abs/2506.09985.

Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mahmoud Assran, and Nicolas Ballas. Revisiting feature prediction for learning visual representations from video. *arXiv preprint arXiv:2404.08471*, 2024. URL https://arxiv.org/abs/2404.08471.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024. URL https://arxiv.org/abs/2410.24164.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1810.12894.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025. URL https://arxiv.org/abs/2506.18088.

Mingtong Dai, Lingbo Liu, Yongjie Bai, Yang Liu, Zhouxia Wang, Rui Su, Chunjie Chen, Liang Lin, and Xinyu Wu. RoVer: Robot reward model as test-time verifier for vision-language-action model. *arXiv preprint arXiv:2510.10975*, 2025. URL https://arxiv.org/abs/2510.10975.

Jacopo Diquigiovanni, Matteo Fontana, and Simone Vantini. Conformal prediction bands for multivariate functional data. *Journal of Multivariate Analysis*, 189:104879, 2022. doi: 10.1016/j.jmva.2021.104879. URL https://doi.org/10.1016/j.jmva.2021.104879.

Jacopo Diquigiovanni, Matteo Fontana, and Simone Vantini. The importance of being a band: Finite-sample exact distribution-free prediction sets for functional data. *Statistica Sinica*, 35(2): 853–871, 2025. doi: 10.5705/ss.202022.0087. URL https://doi.org/10.5705/ss.202022.0087.

Maximilian Du and Shuran Song. DynaGuide: Steering diffusion policies with active dynamic guidance. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, and Yijie Guo. AHA: A vision-language-model for detecting and reasoning over failures in robotic manipulation. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=JVkdSi7Ekg.

Evo-RL Contributors. Evo-RL: Towards iterative policy improvement in real-world offline RL. Software repository, 2026. URL https://github.com/MINT-SJTU/Evo-RL.

Alec Farid, David Snyder, Allen Z. Ren, and Anirudha Majumdar. Failure prediction with statistical guarantees for vision-based robot control. In *Proceedings of Robotics: Science and Systems*, New York City, NY, USA, June 2022. doi: 10.15607/RSS.2022.XVIII.042. URL https://doi.org/10.15607/RSS.2022.XVIII.042.

Youhe Feng, Hansen Shi, Haoyang Li, Xinlei Guo, Yang Wang, Chengyang Zhang, Jinkai Zhang, Xiaohan Zhang, Jie Tang, and Jing Zhang. ProcVLM: Learning procedure-grounded progress rewards for robotic manipulation. *arXiv preprint arXiv:2605.08774*, 2026. URL https://arxiv.org/abs/2605.08774.

Qiao Gu, Yuanliang Ju, Shengxiang Sun, Igor Gilitschenski, Haruki Nishimura, Masha Itkina, and Florian Shkurti. SAFE: Multitask failure detection for vision-language-action models. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1337. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/392d0d05e2f514063e6ce6f8b370834c-Abstract-Conference.html.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4651–4664. PMLR, 2021.

Suhyeok Jang, Dongyoung Kim, Changyeon Kim, Youngsuk Kim, and Jinwoo Shin. Verifier-free test-time sampling for vision-language-action models. In *International Conference on Learning Representations*, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

Jacky Kwok, Christopher Agia, Rohan Sinha, Matt Foutter, Shulu Li, Ion Stoica, Azalia Mirhoseini, and Marco Pavone. RoboMonkey: Scaling test-time sampling and verification for vision-language-action models. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 3200–3217. PMLR, 2025. URL https://proceedings.mlr.press/v305/kwok25a.html.

Sung-Wook Lee, Xuhui Kang, and Yen-Ling Kuo. Diff-DAgger: Uncertainty estimation with diffusion policy for robotic manipulation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4845–4852, 2025. doi: 10.1109/ICRA55743.2025.11127730. URL https://doi.org/10.1109/ICRA55743.2025.11127730.

Tony Lee, Andrew Wagenmaker, Karl Pertsch, Percy Liang, Sergey Levine, and Chelsea Finn. RoboReward: General-purpose vision-language reward models for robotics. *arXiv preprint arXiv:2601.00675*, 2026. URL https://arxiv.org/abs/2601.00675.

Yishu Li, Xinyi Mao, Ying Yuan, Kyutae Sim, Ben Eisner, and David Held. Learn from what we HAVE: History-aware VErifier that reasons about past interactions online. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 2063–2092. PMLR, 2025. URL https://proceedings.mlr.press/v305/li25e.html.

Anthony Liang, Yigit Korkmaz, Jiahui Zhang, Minyoung Hwang, Abrar Anwar, Sidhant Kaushik, Aditya Shah, Alex S. Huang, Luke Zettlemoyer, Dieter Fox, Yu Xiang, Anqi Li, Andreea Bobu, Abhishek Gupta, Stephen Tu, Erdem Biyik, and Jesse Zhang. RoboMeter: Scaling general-purpose robotic reward models via trajectory comparisons. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, 2026. doi: 10.15607/RSS.2026.XXII.140. URL https://www.roboticsproceedings.org/rss22/p140.html.

Zijun Lin, Jiafei Duan, Haoquan Fang, Dieter Fox, Ranjay Krishna, Cheston Tan, and Bihan Wen. FailSafe: Reasoning and recovery from failures in vision-language-action models. *arXiv preprint arXiv:2510.01642*, 2025. URL https://arxiv.org/abs/2510.01642.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 44776–44791, 2023.

Huihan Liu, Shivin Dass, Roberto Martín-Martín, and Yuke Zhu. Model-based runtime monitoring with interactive imitation learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4154–4161. IEEE, 2024. doi: 10.1109/ICRA57147.2024.10611038. URL https://doi.org/10.1109/ICRA57147.2024.10611038.

Huihan Liu, Yu Zhang, Vaarij Betala, Evan Zhang, James Liu, Crystal Ding, and Yuke Zhu. Multi-task interactive robot fleet learning with visual world models. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 4286–4313. PMLR, 2025. URL https://proceedings.mlr.press/v270/liu25g.html.

NVIDIA, Aditi, Niket Agarwal, Arslan Ali, Jon Allen, et al. Cosmos 3: Omnimodal world models for physical AI. *arXiv preprint arXiv:2606.02800*, 2026. URL https://arxiv.org/abs/2606.02800.

Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba, Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023. URL https://arxiv.org/abs/2304.07193.

Paul Pacaud, Ricardo Garcia Pinel, Shizhe Chen, and Cordelia Schmid. Guardian: Detecting robotic planning and execution errors with vision-language models. In *Workshop on Making Sense of Data in Robotics: Composition, Curation, and Interpretability at Scale at CoRL 2025*, 2025. URL https://openreview.net/forum?id=wps46mtC9B.

Seongheon Park, Wendi Li, Changdae Oh, Samuel Yeh, Zsolt Kira, Michael Hagenow, and Sharon Li. Hide-and-Seek in trajectories: Discovering failure signals for VLA runtime monitoring. *arXiv preprint arXiv:2605.30834*, 2026. URL https://arxiv.org/abs/2605.30834.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. URL https://arxiv.org/abs/2504.16054.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763. PMLR, 2021. URL https://proceedings.mlr.press/v139/radford21a.html.

Ziyang Rao, Yiren Zhao, Weiyu Guo, Ben Fei, Yandong Guo, and Hui Xiong. The geometric nature and a free proxy for flow-matching uncertainty. *arXiv preprint arXiv:2607.27933*, 2026. URL https://arxiv.org/abs/2607.27933v1.

Ralf Römer, Adrian Kobras, Luca Worbis, and Angela P. Schoellig. Failure prediction at runtime for generative robot policies. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0260. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/0b7cb3b8cc44e652761245537027db44-Abstract-Conference.html.

Rohan Sinha, Amine Elhafsi, Christopher Agia, Matthew Foutter, Edward Schmerling, and Marco Pavone. Real-time anomaly detection and reactive planning with large language models. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.114. URL https://doi.org/10.15607/RSS.2024.XX.114.

Sruthi Sudhakar, Junbang Liang, Sreehari Rammohan, Pavel Tokmakov, Richard Zemel, and Carl Vondrick. Robot critics that sweat the small stuff. *arXiv preprint arXiv:2606.21572*, 2026. URL https://arxiv.org/abs/2606.21572.

Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*. Springer, New York, NY, 2005. doi: 10.1007/b106715. URL https://doi.org/10.1007/b106715.

Josiah Wong, Albert Tung, Andrey Kurenkov, Ajay Mandlekar, Li Fei-Fei, Silvio Savarese, and Roberto Martín-Martín. Error-aware imitation learning from teleoperation data for mobile manipulation. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 1367–1378. PMLR, 2022. URL https://proceedings.mlr.press/v164/wong22a.html.

Yilin Wu, Ran Tian, Gokul Swamy, and Andrea Bajcsy. From foresight to forethought: VLM-in-the-loop policy steering via latent alignment. In *Proceedings of Robotics: Science and Systems*, June 2025. doi: 10.15607/RSS.2025.XXI.076. URL https://doi.org/10.15607/RSS.2025.XXI.076.

Yilin Wu, Anqi Li, Tucker Hermans, Fabio Ramos, Andrea Bajcsy, and Claudia Pérez-D'Arpino. Do what you say: Steering vision-language-action models via runtime reasoning-action alignment verification. In *2026 IEEE International Conference on Robotics and Automation (ICRA)*, 2026. URL https://arxiv.org/abs/2510.16281.

Chen Xu, Tony Khuong Nguyen, Emma Dixon, Christopher Rodriguez, Patrick Miller, Robert Lee, Paarth Shah, Rares Andrei Ambrus, Haruki Nishimura, and Masha Itkina. Can we detect failures without failure data? uncertainty-aware runtime failure detection for imitation learning policies. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2025. doi: 10.15607/RSS.2025.XXI.073. URL https://doi.org/10.15607/RSS.2025.XXI.073.

Siyuan Yang, Yang Zhang, Haoran He, Ling Pan, Xiu Li, Chenjia Bai, and Xuelong Li. Steering vision-language-action models as anti-exploration: A test-time scaling approach. *arXiv preprint arXiv:2512.02834*, 2025. URL https://arxiv.org/abs/2512.02834.

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-WAM: Do world action models need test-time future imagination? *arXiv preprint arXiv:2603.16666*, 2026. URL https://arxiv.org/abs/2603.16666.

Xianchao Zeng, Xinyu Zhou, Youcheng Li, Jiayou Shi, Tianle Li, Liangming Chen, Lei Ren, and Yong-Lu Li. Diagnose, correct, and learn from manipulation failures via visual symbols. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 42386–42395, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Zeng_Diagnose_Correct_and_Learn_from_Manipulation_Failures_via_Visual_Symbols_CVPR_2026_paper.html.

Haoran Zhang, Yifu Lu, Boyang Wang, Xuhui Kang, Yen-Ling Kuo, Zezhou Cheng, Mengdi Wang, and Odest Chadwicke Jenkins. Foresight: Failure detection for long-horizon robotic manipulation with action-conditioned world model latents. *arXiv preprint arXiv:2606.23085*, 2026. URL https://arxiv.org/abs/2606.23085.

Gehan Zheng, Matthew Johnson-Roberson, and Weiming Zhi. ContactGuard: Pre-contact execution monitoring with action-conditioned latent world models. *arXiv preprint arXiv:2608.13438*, 2026. URL https://arxiv.org/abs/2608.13438.

Shijie Zhou, Bin Zhu, Jiarui Yang, Xiangyu Zhao, Jingjing Chen, and Yu-Gang Jiang. RC-NF: Robot-conditioned normalizing flow for real-time anomaly detection in robotic manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 43050–43060, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Zhou_RC-NF_Robot-Conditioned_Normalizing_Flow_for_Real-Time_Anomaly_Detection_in_Robotic_CVPR_2026_paper.html.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

A Comparison with Model-Based Predictive Monitors

Liu et al. (2024) combine latent-dynamics prediction, OOD detection, and learned failure classification within interactive imitation learning, using human intervention states to supervise failure detection. Their follow-up work, Sirius-Fleet (Liu et al., 2025), extends predictive monitoring to multi-task robot fleets, using visual world models and anomaly predictors to guide human intervention. Concurrent works, Foresight (Zhang et al., 2026) and ContactGuard (Zheng et al., 2026), also assess the anticipated consequences of proposed actions. Below, we clarify how ReAP relates to these approaches.

Failure supervision and prediction space. Foresight freezes a video-pretrained V-JEPA 2 encoder (Assran et al., 2025) and learns an action-conditioned predictor before training a downstream failure detector. ContactGuard learns a JEPA-style world model from unlabelled robot trajectories, then freezes it and fits a linear failure probe. In ReAP, failure supervision first shapes a compact representation on top of a frozen visual backbone. We then freeze the detector and learn action-conditioned transitions within this representation space, using both latent-transition and future-risk supervision. The same frozen detector head scores observed and predicted representations. Thus, failure supervision determines the target space for prediction as well as the subsequent risk readout.

Monitoring scope and temporal context. Foresight aggregates predicted-latent tokens from the execution history with a causal Transformer and uses trajectory-level outcome labels. ContactGuard uses labelled grasp clips and a gripper-closure trigger to assess predicted post-contact states before contact occurs. ReAP also learns from episode-level outcomes, while combining current-observation and predicted-state alarms throughout execution. Its observation branch can assess incoming images independently of action generation, and its prediction branch evaluates proposed action chunks. This design requires neither a contact-event trigger nor a temporal aggregation head.

Runtime considerations. Foresight reports 183.64 ms per monitoring call with its Transformer head on an NVIDIA H200 GPU (Zhang et al., 2026, Table 14). ContactGuard reports 19.18 ms for encoding, a 30-step latent rollout, and classification on an NVIDIA RTX 5090 (Zheng et al., 2026, Table 4). Both reported latencies exclude policy inference. In our setup, the detector takes 9.72 ms, while predictor-based monitoring takes 81.16 ms including policy inference on an NVIDIA A800 80GB GPU (Table 2b). Differences in hardware, prediction horizons, and timing boundaries prevent a controlled speed comparison. Our measurements support the practicality of a fast observation branch that can issue alarms without waiting for policy inference.

Implications for generalization. Foresight's video pretraining may provide temporal and physical priors that help transfer to new settings. Our representation learns failure-related cues from labeled rollouts, so its transfer performance may depend on the range of tasks and failure modes covered during training. Evaluating both approaches under the same task and policy shifts would help establish how these differences affect their generalization.

B Comparison with Vision-Based Monitors

Several recent failure monitors assess robot execution directly from visual observations or visually grounded representations. VLM-based approaches exploit semantic reasoning to recognize and interpret manipulation failures. AHA (Duan et al., 2025) formulates failure detection as free-form visual reasoning, while Guardian (Pacaud et al., 2025) learns detailed failure detection and reasoning from automatically synthesized failure data. FailSafe (Lin et al., 2025) extends visual failure reasoning toward recovery, and ViFailback (Zeng et al., 2026) uses real-world failure trajectories to train a VLM for failure diagnosis and visually grounded correction guidance. SENTINEL (Agia et al., 2025) incorporates a VLM-based progress monitor alongside a separate action-consistency signal for runtime failure detection. More broadly, recent progress and reward models learn from large-scale robot trajectory datasets to assess task progress and execution quality across successful and unsuccessful rollouts (Liang et al., 2026; ACE-Brain Team et al., 2026; Feng et al., 2026; Lee et al., 2026).

Table 5: Comparison with RoboMeter on LIBERO-10. Latency denotes standalone inference time on a single NVIDIA A800 80GB GPU.

	OpenVLA		π_0		
Method	bTWA ↑	bACC ↑	bTWA ↑	bACC ↑	Latency (ms) ↓
RoboMeter	0.669	0.739	0.714	0.831	109.40
ReAP (Detector)	**0.671**	**0.744**	**0.750**	**0.838**	**9.72**

Beyond large VLMs, observation-side monitoring has also been studied with lighter learned models. RC-NF (Zhou et al., 2026), for example, combines object-motion cues extracted from visual observations with robot state in a robot-conditioned normalizing flow for real-time anomaly detection. Many existing vision-based monitors rely on multi-billion-parameter vision-language models, and some additionally require explicit semantic reasoning, leading to substantially higher computational cost than lightweight per-step monitors. RC-NF (Zhou et al., 2026) is designed for real-time monitoring, but its official implementation was not publicly available at the time of our experiments. We therefore choose RoboMeter (Liang et al., 2026) for quantitative comparison, as it has relatively low inference latency, is pretrained on large-scale robot trajectory data, and its learned progress signal can be directly adapted to failure detection.

We fine-tune RoboMeter on the full training split of our collected rollouts and adapt its progress prediction for frame-level failure monitoring. At timestep t, we sample eight frames from the observed trajectory prefix and use RoboMeter to estimate the current task progress p_t, performing inference every four environment frames. We then calibrate a time-varying lower band $L_\alpha(t)$ from successful progress trajectories and raise an alarm when $p_t \leq L_\alpha(t)$. Threshold selection and evaluation follow the same protocol as our ReAP.

As shown in Table 5, our observation-only detector achieves higher bTWA and bACC than RoboMeter in both LIBERO-10 policy settings, while requiring substantially lower standalone inference latency. We also observe failed rollouts for which RoboMeter assigns relatively high progress scores, suggesting that progress estimation may not always capture fine-grained visual failure cues. This observation is consistent with recent VLM-critic work showing that distinguishing fine-grained visual differences between successful and failed executions remains challenging for vision-language models (Sudhakar et al., 2026).

C Implementation Details

C.1 Benchmark Details

We construct disjoint training, validation, calibration, and evaluation splits for each policy–benchmark setting, as summarized in Table 6. The training split is used to train the detector and predictor, while the validation split is additionally collected for hyperparameter and model selection. The calibration split contains held-out successful rollouts used only to fit the monitoring thresholds after model selection. All reported metrics are computed on the separate evaluation split. The RoboTwin-8 task subsets are selected separately for $\pi_{0.5}$ and Fast-WAM according to their policy success rates. The selected tasks and their corresponding success rates are reported in Table 7.

For functional threshold calibration, we randomly divide the successful calibration trajectories into two subsets using a 30/70 split, following the functional calibration implementation of SAFE (Gu et al., 2025). The first subset is used to estimate the mean trajectory and temporal modulation function, while the second subset calibrates the width of the conformal band. The detailed threshold construction is described in Appendix E.

Our timestep supervision requires only the final success or failure outcome of each rollout. Successful trajectories are assigned zero risk throughout execution, while failed trajectories use the temporally increasing target $y_t = t/(T-1)$ defined in Section 3. The linear target is a weak trajectory-level risk proxy rather than a ground-truth failure-onset annotation. Compared with assigning the same binary failure label to every frame of a failed rollout, it avoids treating early, often nominal-looking

Table 6: Dataset splits used for training, validation, threshold calibration, and evaluation. Validation rollouts are used for hyperparameter and model selection, while calibration contains held-out successful rollouts used only to fit the monitoring thresholds.

Setting	Train	Validation	Calibration	Evaluation
LIBERO-10	300	50	50	150
RoboTwin-8	192	32	32	96
Fold Cloth	200	50	50	40
Stack Plates	42	22	20	40
Pick Cubes	40	23	20	40

Table 7: Selected RoboTwin-8 tasks and corresponding base-policy success rates. The task subsets are selected separately for $\pi_{0.5}$ and Fast-WAM.

(a) $\pi_{0.5}$

Task instruction	Success Rate
Lift the pot with both arms.	25.0%
Pick up one bottle with each arm.	27.5%
Pick up the hamburger and fries and place them on the tray.	62.5%
Place the container onto the plate.	72.5%
Pick up the playing card and move it away from the table.	70.0%
Press the top button of the alarm clock.	72.5%
Pick up the bread and place it into the skillet.	20.0%
Place the empty cup on the coaster.	37.5%

(b) Fast-WAM

Task instruction	Success Rate
Grasp the red block with the left arm, hand it to the right arm, and place it on the blue pad.	67.5%
Pick up the mug, reorient it, and hang it on the rack.	60.0%
Move the stapler to a colored mat.	52.5%
Open the microwave with one arm.	55.0%
Pick up one bottle with each arm.	67.5%
Place the can into the basket and lift the basket.	72.5%
Stack the three bowls on top of each other.	82.5%
Press the switch.	75.0%

frames as equally erroneous. We empirically found this linear supervision to perform better than binary targets for both ReAP and the supervised SAFE baselines in Appendix D.2.

C.2 Baseline Implementations

For RND, LogpZO, SAFE-MLP, and SAFE-LSTM, we build on the public SAFE implementation[1] and follow its model architectures and optimization hyperparameters unless otherwise specified. RND and LogpZO are trained only on successful rollouts and therefore do not use failure-risk labels. For the supervised SAFE baselines, we replace the binary failure target with the same temporally increasing target $y_t = t/(T-1)$ used by our detector as discussed above.

For baselines that operate on policy representations, we follow SAFE for feature extraction whenever applicable. For OpenVLA, we mean-pool the hidden states corresponding to the seven autoregressive action tokens, yielding a 4096-D representation. For π_0 and $\pi_{0.5}$, we average action-expert suffix hidden states across the action horizon and ten flow-matching steps to obtain a 1024-D representation. For Fast-WAM, we mean-pool the final video-MoT policy tokens into a 3072-D representation.

STAC, Diff-DAgger, ACCEL, and FIPER are implemented following their respective public implementations. For methods that require repeated action sampling, we generate 32 action chunks at

[1] https://github.com/vla-safe/SAFE

Table 8: Architecture and training configuration of ReAP. The same configuration is used across policy–benchmark settings unless otherwise specified.

Architecture		Training	
Setting	Value	Setting	Value
Visual backbone	DINOv2-Large	Optimizer	AdamW
Latent queries M	32	Batch size	64
Detector blocks N_d	2	Training steps	10k
Detector latent width	1024	Learning rate	1×10^{-4}
Predictor blocks N_p	7	Weight decay	0.01
State / predictor width	256	$(\lambda_{\text{feat}}, \lambda_{\text{risk}})$	$(1, 1)$
Prediction horizon H	1 (OpenVLA) 10 (others)	LR schedule	Cosine 500-step warmup

each replanning step. We retain the sampled action chunks and intermediate flow trajectories required by the corresponding baselines. For Fast-WAM, repeated-sampling methods likewise use 32 action samples with an action horizon of 32 and ten inference steps.

OpenVLA requires separate treatment because its standard rollout policy uses deterministic autoregressive decoding. We apply this sampling procedure only to FIPER, whose action-uncertainty component can be constructed directly from a set of stochastic action proposals. Diff-DAgger and ACCEL require policy-specific diffusion or flow-matching quantities that are not defined for OpenVLA. STAC is not applicable to OpenVLA because it requires temporally overlapping action chunks across consecutive policy queries, whereas OpenVLA predicts only a single immediate action per query. We evaluate FIPER on OpenVLA using stochastic action proposals following the sampling strategy of RoboMonkey (Kwok et al., 2025). Specifically, we first obtain multiple valid action samples through stochastic autoregressive decoding, then fit a diagonal Gaussian to the first six continuous action dimensions, and draw additional proposals from the fitted distribution. The gripper action is determined by majority vote, and continuous actions are clipped to the policy action-statistics range. This sampling procedure is used only to construct the FIPER uncertainty signal and does not alter the deterministic OpenVLA rollouts used for evaluation.

FIPER uses the thresholding procedure from its original implementation. For the remaining baselines, we select the time-varying calibration variant using the validation split and fit its thresholds on the held-out successful calibration rollouts using our six calibration variants.

C.3 Architecture and Training Details

We use the same monitor architecture and training configuration across all policy–benchmark settings unless otherwise stated. The prediction horizon H is measured in environment control steps. Each training sample pairs the current observation O_t with the H actions executed before the future observation O_{t+H}. At inference, the predictor evaluates the first H actions proposed by the current policy query. Simulation experiments train one monitor jointly across all tasks within each setting, while each real-world task uses a separately trained monitor. The main architecture and training configuration is summarized in Table 8. We use the LeRobot dataset interface[2] to load our collected rollouts and sample training data. Image augmentation, including random cropping and small rotations for third-person views and color jitter for all camera views, is applied during training. Each detector and predictor is trained on a single NVIDIA A800 80GB GPU.

C.4 Real-Robot Setup

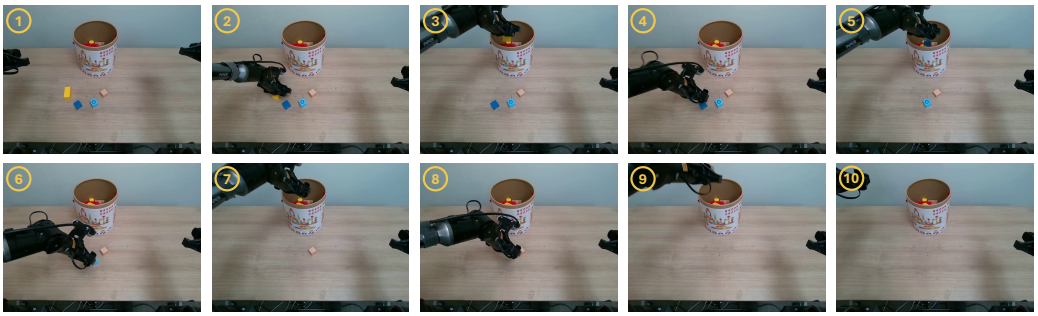
Our real-world experiments use a dual-arm robotic platform consisting of two AgileX PIPER manipulators. Visual observations are captured by three Intel RealSense D435i cameras, including one global third-person camera and one wrist-mounted camera on each arm. For each manipulation task, we deploy a separately fine-tuned $\pi_{0.5}$ policy and train a task-specific ReAP monitor using the same model architecture and training configuration as in simulation.

[2] https://github.com/huggingface/lerobot

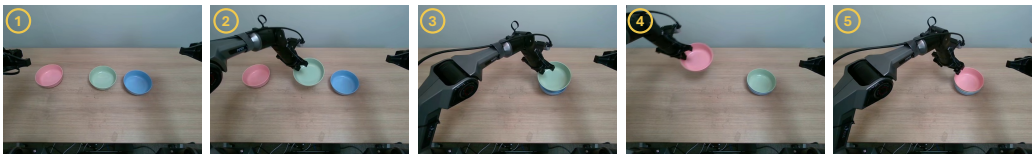
Our deployment system is built on the Evo-RL framework (Evo-RL Contributors, 2026), which we extend to support runtime failure monitoring. Observation acquisition, policy inference, and robot control run in separate processes. A dedicated observation server continuously provides the latest synchronized camera observations. The control process retrieves the latest observation and sends it to a remote inference server, where the deployed policy generates the next action chunk. The predictor runs alongside the policy on the inference server and evaluates each proposed action chunk before execution, while the observation-based detector runs independently on a local NVIDIA RTX 4070 GPU and monitors incoming observations without waiting for policy inference. The action chunk returned by the server is then executed by the robot. Whenever either branch raises an alarm, we record the alarm together with the corresponding rollout information for subsequent evaluation. ReAP is the only monitor integrated into the live deployment system. Baseline predictions are generated post hoc from the same recorded observations and policy outputs. Since monitoring alarms do not alter robot execution, all methods are evaluated on identical successful and failed trajectories, enabling paired comparison without collecting separate physical rollouts for each baseline.



(a) Fold Cloth.



(b) Pick Cubes.



(c) Stack Plates.

Figure 5: Representative real-world rollouts across three manipulation tasks.

Figure 5 illustrates the three real-world manipulation tasks. In Fold Cloth, the two arms coordinate to lift, reposition, and repeatedly fold a horizontally placed garment. In Stack Plates, the robot manipulates three dishes and stacks them into a single pile. In Pick Cubes, the robot picks four cubes from the tabletop and places them into a container.

D Additional Analyses

D.1 Generalization Analyses

Seen-to-unseen generalization. A useful failure representation should preserve failure-relevant cues as task appearance and interaction context change (Gu et al., 2025). We evaluate this property on OpenVLA by randomly selecting three LIBERO-10 tasks (Task 1, 4, 7) as unseen tasks, training all methods on the remaining seen tasks, calibrating corresponding thresholds on unseen tasks, and evaluating them on a disjoint set of unseen-task rollouts without further model training. As shown in Table 9, performance on unseen tasks is relatively close across methods, reflecting the difficulty of transferring failure-monitoring signals across task distributions. ReAP achieves the highest mean performance, reaching 0.568 bTWA and 0.677 bACC. This suggests that the failure-supervised representation captures failure cues that remain useful beyond the tasks observed during training. However, the margin over the strongest baselines is modest, indicating that cross-task transfer remains challenging and that the learned failure representation is still affected by task-specific visual and interaction patterns.

Table 9: Seen-to-unseen failure monitoring results.

Method	bTWA ↑	bACC ↑
SAFE-MLP	$0.537^{\pm 0.002}$	$0.653^{\pm 0.007}$
SAFE-LSTM	$0.556^{\pm 0.093}$	$0.632^{\pm 0.090}$
LogpZO	$0.523^{\pm 0.070}$	$0.592^{\pm 0.089}$
RND	$0.552^{\pm 0.030}$	$0.661^{\pm 0.034}$
FIPER	$0.548^{\pm 0.000}$	$0.600^{\pm 0.000}$
ReAP (Ours)	$\mathbf{0.568}^{\pm 0.064}$	$\mathbf{0.677}^{\pm 0.070}$

Table 10: Cross-policy failure monitoring on LIBERO-10.

Method	Transfer	bTWA ↑	bACC ↑
RoboMeter	OpenVLA $\rightarrow \pi_0$	0.685	0.786
RoboMeter	$\pi_0 \rightarrow$ OpenVLA	**0.630**	**0.685**
ReAP	OpenVLA $\rightarrow \pi_0$	**0.746**	**0.818**
ReAP	$\pi_0 \rightarrow$ OpenVLA	0.629	0.676

Cross-policy generalization. We next evaluate whether ReAP transfers across policies on LIBERO-10. Since ReAP relies on external visual observations and proposed action chunks without accessing policy-internal representations, the same monitor can be applied to a different policy without architectural modification. We train and calibrate the monitor using rollouts from one source policy and directly evaluate it on another target policy without additional adaptation. For each transfer direction, we train the monitor on source-policy rollouts using the prediction horizon available at target-policy evaluation. For OpenVLA $\rightarrow \pi_0$, we construct ten-step action sequences from consecutive actions in the recorded OpenVLA rollouts and train the predictor with $H = 10$. At evaluation, we use the first ten actions of each action chunk proposed by π_0. For $\pi_0 \rightarrow$ OpenVLA, we train and evaluate with $H = 1$, matching OpenVLA's single-step action proposal.

As shown in Table 10, OpenVLA $\rightarrow \pi_0$ achieves 0.746 bTWA, compared with 0.780 under π_0 in-policy training in Table 1, while $\pi_0 \rightarrow$ OpenVLA reaches 0.629 compared with 0.701 under the corresponding OpenVLA in-policy setting. Although performance decreases after transferring across policies, it remains relatively close to the corresponding in-policy results in both settings. This shows that ReAP can preserve useful failure-monitoring capability across different deployed policies without target-policy adaptation. We further compare cross-policy transfer with RoboMeter, since it is likewise independent of policy-internal signals. When transferring to π_0, ReAP achieves 0.746 bTWA and 0.818 bACC, compared with 0.685 and 0.786 for RoboMeter. When transferring to OpenVLA, the two methods perform comparably, with ReAP obtaining 0.629 bTWA and 0.676 bACC and RoboMeter obtaining 0.630 and 0.685. Across the two transfer settings, ReAP therefore maintains a relatively stable advantage or comparable performance against RoboMeter.

D.2 Additional Component Ablations

Visual backbone. To examine whether the detector depends on a particular pretrained visual representation, we replace the visual backbone while keeping the detector architecture and training protocol unchanged. Table 11 shows relatively stable performance across CLIP (Radford et al., 2021) and different DINOv2 variants, suggesting that the detector is not highly sensitive to the

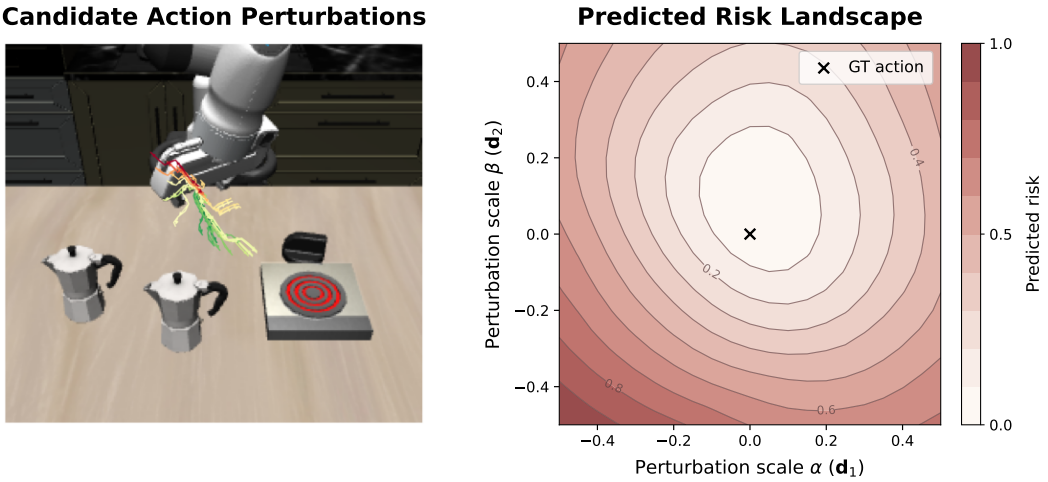


Figure 6: Action-sensitivity visualization on π_0/LIBERO-10 under controlled perturbations of a candidate action chunk.

choice of pretrained visual encoder. DINOv2-Small achieves the highest mean performance, reaching 0.672 bTWA and 0.760 bACC, while the default DINOv2-Large obtains a nearly identical bTWA of 0.671. Performance remains relatively stable across backbone choices, suggesting that ReAP can effectively use features from different pretrained visual backbones.

Table 11: Visual backbone ablation on OpenVLA/LIBERO-10.

Visual Backbone	bTWA ↑	bACC ↑	T-det ↓
CLIP ViT-B/32	$0.654^{\pm 0.026}$	$0.740^{\pm 0.030}$	$0.462^{\pm 0.062}$
DINOv2-Small	$\mathbf{0.672}^{\pm 0.025}$	$\mathbf{0.760}^{\pm 0.017}$	$\mathbf{0.432}^{\pm 0.059}$
DINOv2-Base	$0.657^{\pm 0.013}$	$0.747^{\pm 0.016}$	$0.450^{\pm 0.043}$
DINOv2-Large	$0.671^{\pm 0.021}$	$0.744^{\pm 0.020}$	$0.450^{\pm 0.041}$

Table 12: Ablation of the prediction horizon on π_0/LIBERO-10.

Prediction Horizon	bTWA ↑	bACC ↑	T-det ↓
$H = 1$	$0.745^{\pm 0.002}$	$0.834^{\pm 0.014}$	$0.464^{\pm 0.027}$
$H = 5$	$0.744^{\pm 0.004}$	$\mathbf{0.837}^{\pm 0.010}$	$0.464^{\pm 0.011}$
$\boldsymbol{H = 10}$	$\mathbf{0.755}^{\pm 0.009}$	$0.831^{\pm 0.010}$	$\mathbf{0.461}^{\pm 0.017}$
$H = 20$	$0.736^{\pm 0.002}$	$0.826^{\pm 0.008}$	$0.475^{\pm 0.021}$
$H = 30$	$0.736^{\pm 0.004}$	$0.829^{\pm 0.005}$	$0.479^{\pm 0.018}$

Prediction horizon. The prediction horizon H determines how far ahead the predictor estimates the consequences of a proposed action chunk. We evaluate horizons from $H = 1$ to $H = 30$. As shown in Table 12, performance does not improve monotonically with the prediction horizon. $H = 5$ achieves the highest bACC of 0.837, while $H = 10$ achieves the highest bTWA of 0.755 and the numerically lowest T-det of 0.461. Increasing the horizon to $H = 20$ or $H = 30$ reduces bTWA to 0.736 and results in somewhat later alarms. We therefore use $H = 10$ as the default horizon for chunked-action policies, based on its overall balance of detection accuracy and timeliness.

Action Sensitivity. To qualitatively examine how the predictor responds to changes in the candidate action, we fix an observation from π_0/LIBERO-10 and visualize its predictions in Fig. 6. The left panel shows candidate action trajectories around the current gripper pose. We take a reference action chunk $\mathbf{a}_0$ from a held-out successful rollout and perturb it along two temporally smooth and orthogonal directions $\mathbf{d}_1$ and $\mathbf{d}_2$, forming $\mathbf{a}_{\alpha,\beta} = \mathbf{a}_0 + \alpha\mathbf{d}_1 + \beta\mathbf{d}_2$. The right panel shows the resulting predicted risk landscape, with lower risk near the reference action and generally higher risk for larger deviations. This visualization illustrates the predictor's sensitivity to action deviations under a fixed observation, complementing the quantitative action-conditioning results in Table 3b and Table 4.

Effect of Training Labels We compare binary and temporally increasing risk targets on OpenVLA/LIBERO-10. The binary target assigns a risk label of one to every observation in a failed rollout, while our default target increases linearly from zero to one over that rollout. Successful rollouts receive a target of zero in both cases. Table 13 shows that the effect of the label depends on the monitored model and the evaluation metric. Although binary labels yield higher

Table 13: Effect of training labels on OpenVLA/LIBERO-10.

Model	Label	bTWA ↑	bACC ↑	T-det ↓
SAFE-MLP	Binary	$0.498^{\pm 0.005}$	$\mathbf{0.797}^{\pm 0.013}$	$0.925^{\pm 0.009}$
SAFE-MLP	Temporal	$0.622^{\pm 0.004}$	$0.687^{\pm 0.001}$	$0.641^{\pm 0.019}$
ReAP (Detector)	Binary	$0.465^{\pm 0.020}$	$0.669^{\pm 0.044}$	$0.946^{\pm 0.034}$
ReAP (Detector)	Temporal	$\mathbf{0.671}^{\pm 0.021}$	$0.744^{\pm 0.020}$	$\mathbf{0.450}^{\pm 0.041}$

Table 14: Detection timing and failure coverage on LIBERO-10. Detected failures reports the mean number of correctly detected failures out of 75 for OpenVLA and 60 for π_0. Positive Δt indicates alarms before the annotated failure onset, while negative values indicate delayed alarms. Δt is computed only over correctly detected failed rollouts.

	OpenVLA			π_0		
Method	Detected failures ↑	Lag Δt ↑	bACC ↑	Detected failures ↑	Lag Δt ↑	bACC ↑
STAC	–	–	–	$56.4^{\pm 0.0}$	$+0.076^{\pm 0.000}$	$0.578^{\pm 0.000}$
Diff-DAgger	–	–	–	$\mathbf{59.4}^{\pm 0.0}$	$\mathbf{+0.120}^{\pm 0.000}$	$0.531^{\pm 0.000}$
ACCEL	–	–	–	$46.4^{\pm 0.0}$	$-0.118^{\pm 0.000}$	$0.777^{\pm 0.000}$
RND	$49.9^{\pm 10.3}$	$-0.138^{\pm 0.038}$	$0.715^{\pm 0.048}$	$50.3^{\pm 0.8}$	$-0.044^{\pm 0.023}$	$\mathbf{0.874}^{\pm 0.010}$
LogpZO	$22.7^{\pm 13.6}$	$-0.074^{\pm 0.059}$	$0.531^{\pm 0.019}$	$50.6^{\pm 0.3}$	$-0.097^{\pm 0.010}$	$0.862^{\pm 0.007}$
FIPER	$51.3^{\pm 5.5}$	$-0.487^{\pm 0.007}$	$\mathbf{0.808}^{\pm 0.035}$	$50.9^{\pm 1.5}$	$-0.177^{\pm 0.021}$	$0.866^{\pm 0.005}$
SAFE-MLP	$36.7^{\pm 2.1}$	$-0.050^{\pm 0.005}$	$0.687^{\pm 0.001}$	$50.6^{\pm 0.3}$	$-0.019^{\pm 0.005}$	$0.869^{\pm 0.005}$
SAFE-LSTM	$12.3^{\pm 10.9}$	$\mathbf{+0.100}^{\pm 0.130}$	$0.528^{\pm 0.037}$	$54.0^{\pm 1.1}$	$-0.057^{\pm 0.014}$	$0.871^{\pm 0.010}$
ReAP (Ours)	$\mathbf{60.9}^{\pm 2.3}$	$+0.081^{\pm 0.011}$	$0.770^{\pm 0.008}$	$46.6^{\pm 2.3}$	$+0.045^{\pm 0.010}$	$0.865^{\pm 0.015}$

bACC for SAFE-MLP, they lead to substantially later alarms and consequently lower bTWA. The ReAP detector trained with binary labels also raises alarms much later and achieves lower bTWA than its temporally supervised counterpart. These results support the temporally increasing target for the accuracy–timeliness trade-off measured by bTWA.

D.3 Alarm Timing Relative to Annotated Failure Onset

To provide a temporal reference for evaluating alarm timing, we use Qwen3.5-9B (Qwen Team, 2026) to obtain approximate failure-onset annotations for failed rollouts on LIBERO-10. We adapt the annotation prompt from Appendix F.1 of Park et al. (2026) and use a coarse-to-fine procedure with two successive model calls per rollout. In both calls, the model receives image grids containing the main-camera and wrist-camera views at each sampled time point, labeled with the frame index and timestamp. The first call locates the approximate onset using frames sampled throughout the rollout every 25 timesteps, including the first and last frames. Each 535-frame failed rollout yields 23 sampled time points, arranged into four image grids. The second call then refines this estimate using denser samples taken every three frames within a window extending 30 frames before and after the initial estimate. These model-generated annotations serve only as an approximate temporal reference for this supplementary analysis and are not used for training or threshold calibration.

We compute the normalized detection lag for each method and calibration level as

$$\Delta t = \frac{1}{|\mathcal{TP}|} \sum_{i \in \mathcal{TP}} \frac{t_i^* - t_{\text{alarm},i}}{T_i}, \tag{5}$$

where $\mathcal{TP}$ denotes correctly detected failed rollouts, t_i^* is the annotated failure onset, $t_{\text{alarm},i}$ is the first alarm time, and T_i is the rollout length. Positive values indicate alarms before the annotated onset, while negative values indicate delayed alarms. Missed failures are excluded from Δt and are reflected in bACC. We report both metrics averaged over $\alpha \in \{0.05, 0.10, 0.15, 0.20, 0.25\}$.

Table 14 highlights the trade-off between warning timeliness, failure coverage, and false alarms. On OpenVLA, SAFE-LSTM's larger mean warning lead is accompanied by limited failure coverage, while FIPER achieves higher bACC with delayed alarms. ReAP detects the most failures while still

Table 15: Threshold-calibration ablation for the detector on LIBERO-10.

	OpenVLA			π_0		
Calibration	bTWA ↑	bACC ↑	T-det ↓	bTWA ↑	bACC ↑	T-det ↓
Normalized functional	$\mathbf{0.671}^{\pm 0.021}$	$\mathbf{0.744}^{\pm 0.020}$	$0.450^{\pm 0.041}$	$\mathbf{0.750}^{\pm 0.005}$	$\mathbf{0.838}^{\pm 0.013}$	$0.466^{\pm 0.014}$
Normalized pointwise	$0.574^{\pm 0.031}$	$0.618^{\pm 0.033}$	$\mathbf{0.147}^{\pm 0.021}$	$0.703^{\pm 0.013}$	$0.790^{\pm 0.008}$	$0.318^{\pm 0.007}$
Absolute functional	$0.535^{\pm 0.010}$	$0.645^{\pm 0.032}$	$0.685^{\pm 0.036}$	$0.690^{\pm 0.034}$	$0.825^{\pm 0.023}$	$0.532^{\pm 0.037}$
Absolute pointwise	$0.486^{\pm 0.020}$	$0.546^{\pm 0.021}$	$0.203^{\pm 0.020}$	$0.711^{\pm 0.008}$	$0.791^{\pm 0.009}$	$\mathbf{0.299}^{\pm 0.008}$
Stage functional	$0.495^{\pm 0.003}$	$0.527^{\pm 0.017}$	$0.982^{\pm 0.010}$	$0.633^{\pm 0.016}$	$0.786^{\pm 0.019}$	$0.704^{\pm 0.042}$
Stage pointwise	$0.522^{\pm 0.004}$	$0.638^{\pm 0.009}$	$0.784^{\pm 0.027}$	$0.657^{\pm 0.014}$	$0.782^{\pm 0.025}$	$0.488^{\pm 0.056}$

Table 16: Comparison of OR and AND fusion rules across four policy–benchmark settings.

Benchmark	Fusion	bTWA ↑	bACC ↑	T-det ↓
OpenVLA / LIBERO-10	**OR**	$\mathbf{0.701}^{\pm 0.010}$	$\mathbf{0.770}^{\pm 0.008}$	$\mathbf{0.326}^{\pm 0.029}$
	AND	$0.653^{\pm 0.035}$	$0.737^{\pm 0.042}$	$0.530^{\pm 0.072}$
π_0 / LIBERO-10	**OR**	$\mathbf{0.780}^{\pm 0.008}$	$\mathbf{0.865}^{\pm 0.015}$	$\mathbf{0.395}^{\pm 0.025}$
	AND	$0.726^{\pm 0.006}$	$0.804^{\pm 0.006}$	$0.532^{\pm 0.013}$
$\pi_{0.5}$ / RoboTwin-8	**OR**	$\mathbf{0.701}^{\pm 0.013}$	$0.785^{\pm 0.011}$	$\mathbf{0.306}^{\pm 0.021}$
	AND	$0.683^{\pm 0.024}$	$\mathbf{0.802}^{\pm 0.023}$	$0.392^{\pm 0.032}$
Fast-WAM / RoboTwin-8	**OR**	$\mathbf{0.651}^{\pm 0.014}$	$\mathbf{0.779}^{\pm 0.008}$	$\mathbf{0.592}^{\pm 0.051}$
	AND	$0.614^{\pm 0.014}$	$0.734^{\pm 0.014}$	$0.725^{\pm 0.036}$

raising alarms before the annotated onset on average. On π_0, baselines that warn earlier than ReAP incur substantially more false alarms, whereas those with slightly higher bACC raise alarms after the annotated onset on average. Across both policies, ReAP combines positive mean warning leads with competitive balanced accuracy, supporting its ability to provide advance warnings while controlling the false-alarm cost.

D.4 Threshold Calibration Analysis

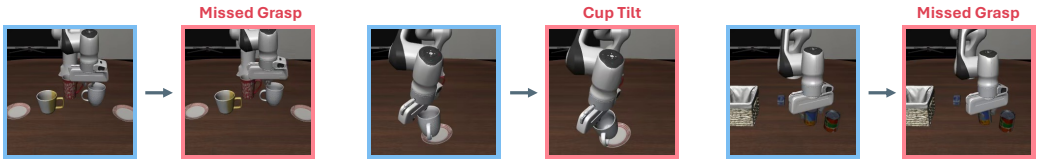
The calibration procedure determines how continuous risk scores are converted into failure alarms and therefore directly affects the trade-off between detection accuracy and warning timeliness. We evaluate six variants obtained by combining three temporal parameterizations with pointwise or functional calibration. As shown in Table 15, normalized-time functional calibration achieves the highest bTWA on both OpenVLA and π_0. Pointwise calibration generally produces earlier alarms, as reflected by lower T-det, but this gain is often accompanied by lower bACC and overall bTWA.

The distinction between pointwise and trajectory-level calibration helps explain these results. As described in Appendix E, pointwise calibration estimates a separate quantile at each temporal position. However, a successful rollout is counted as a false alarm if its score crosses the threshold at any monitored timestep, so controlling exceedances separately at each position does not directly control false alarms over the full rollout. Functional calibration instead uses the maximum standardized deviation along each successful trajectory to determine the band width, explicitly accounting for threshold crossings across time. This provides a possible explanation for the later alarms but higher bACC and bTWA observed with normalized-time functional calibration. Normalized time may further improve calibration by reducing temporal misalignment between rollouts of different durations. However, different monitoring methods may exhibit different temporal patterns in their risk scores, so the most effective calibration variant may vary across methods. We therefore select the calibration variant separately for each method on the validation split before fitting the final thresholds on held-out successful calibration rollouts.

D.5 Fusion Analysis

Table 17: Trajectory-level alarm overlap and fusion outcomes at $\alpha = 0.15$ for seed 2. Each policy has 150 evaluation rollouts, including 75 failures for OpenVLA and 60 for π_0. OR and AND denote the union and intersection of trajectory-level branch alarms, respectively. OR triggers at the first branch alarm, while AND triggers once both branches have alarmed. Alarms on successful rollouts are false alarms.

	OpenVLA			π_0		
Alarm pattern	Total	Failed	Successful	Total	Failed	Successful
Both alarm	47	41	6	39	38	1
Predictor only	24	17	7	5	4	1
Detector only	12	4	8	10	9	1
Neither alarms	67	13	54	96	9	87
OR	83	62	21	54	51	3
AND	47	41	6	39	38	1



(a)



(b)

Figure 7: Qualitative examples of complementary detector and predictor alarms. (a) Predictor-only cases. Blue boxes show the observation when the predictor raises an alarm, while red boxes show a later state where the failure becomes visually apparent. (b) Detector-only cases, where the observed state already contains clear visual failure evidence.

Fusion rule. Table 16 compares OR and AND fusion across four policy–benchmark settings. AND flags a rollout once both branches have raised at least one alarm, with the fused alarm time given by the later of their first-alarm times. If either branch never alarms, AND does not flag the rollout. OR achieves higher bTWA and earlier detection in every setting, together with higher bACC in three of the four settings. Since bTWA explicitly penalizes false alarms on successful rollouts, its consistent improvement indicates that the gains in time-weighted failure detection outweigh the additional false-alarm cost under this metric. OR also outperforms both standalone branches in bTWA across all four settings in Table 1.

To examine the source of this improvement, Table 17 summarizes branch alarm overlap across 150 evaluation rollouts for each LIBERO-10 policy at $\alpha = 0.15$. On OpenVLA, the predictor uniquely detects 17 failed rollouts, while the detector uniquely detects four. On π_0, the corresponding counts are four and nine. By retaining these branch-specific warnings, OR detects 21 additional failed OpenVLA rollouts and 13 additional failed π_0 rollouts compared with AND, while introducing 15 and two additional false alarms, respectively. AND misses failures detected by only one branch, while OR retains these warnings and provides broader failure coverage.

Figure 7 provides qualitative examples of these branch-specific alarms. In predictor-only cases, the current observation can remain visually plausible while the proposed action already points toward an undesirable future outcome, allowing the predictor to raise an alarm before clear failure evidence

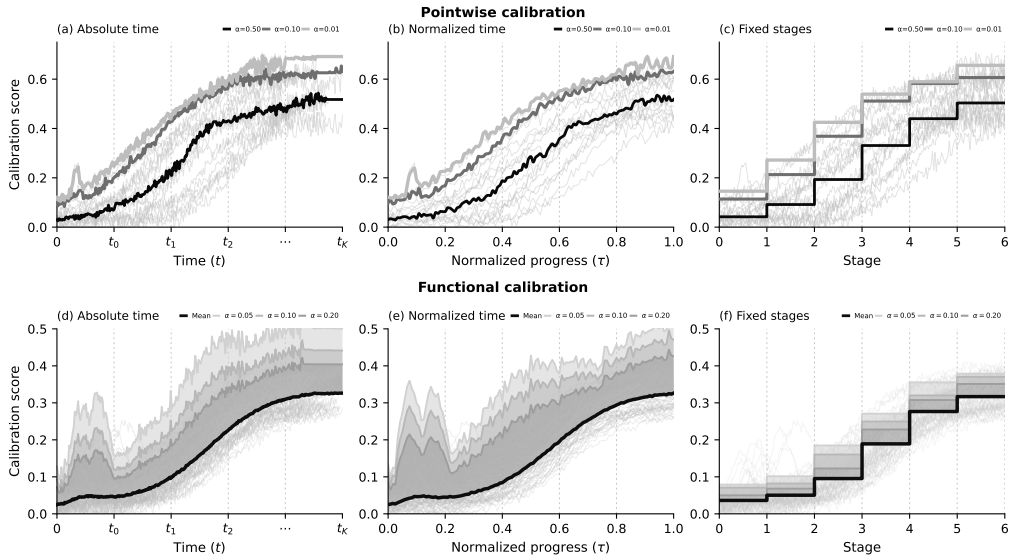


Figure 8: Illustration of the six time-varying threshold calibration variants. The top row shows pointwise empirical-quantile calibration and the bottom row shows functional calibration. From left to right, the columns use absolute time, normalized execution progress, and fixed execution stages. Thin gray curves denote successful calibration trajectories. Dark curves in the top row indicate thresholds at representative significance levels α, while the bottom row shows the nominal trajectory together with the corresponding functional upper bands. Smaller α produces a more conservative upper boundary.

becomes observable. In detector-only cases, the failure has already manifested as a salient visual deviation, such as a tipped object or failed insertion, making it more directly recognizable from the current observation. These examples are consistent with the overlap analysis and illustrate the complementary roles of current-state failure recognition and action-conditioned future-risk prediction.

E Time-Varying Threshold Calibration

We calibrate alarm thresholds using held-out successful rollouts to account for variation in risk scores over the course of execution. For rollout i, let T_i denote its full length in environment frames. A monitoring branch records L_i risk scores, forming the sequence $\mathbf{s}_i = (s_{i,0}, \ldots, s_{i,L_i-1})$. Each score $s_{i,j}$ is associated with an original frame index $f_{i,j}$, and larger scores indicate higher predicted failure risk. For the predictor, $f_{i,j}$ refers to the decision frame at which the proposed action chunk is evaluated.

We calibrate thresholds separately for the detector and predictor. For each branch, let $\mathcal{D}_{\mathrm{cal}} = \{\mathbf{s}_1, \ldots, \mathbf{s}_N\}$ denote the risk-score sequences from held-out successful rollouts. These sequences are used to estimate a time-varying upper threshold $\delta_j(\alpha)$, where j indexes monitored positions under the chosen temporal parameterization. We evaluate $\alpha \in \{0.05, 0.10, 0.15, 0.20, 0.25\}$ and use the same α for both branches. Each branch raises an alarm at the first position satisfying $s_{i,j} \geq \delta_j(\alpha)$, and the two branches' alarms are combined using the OR rule described in the main paper.

A single global threshold ignores temporal variation in the score distribution over the course of an execution. Successful manipulation trajectories can exhibit different nominal risk levels during phases such as free-space motion, contact, grasping, and placement. We therefore consider three temporal parameterizations consisting of normalized time, absolute time, and fixed execution stages. Each temporal parameterization is paired with either pointwise empirical-quantile calibration or functional conformal calibration, resulting in the six variants illustrated in Figure 8.

E.1 Temporal Parameterizations

Absolute time. Absolute-time calibration aligns successful trajectories by their monitored score index j. Let $L_{\max} = \max_i L_i$ be the maximum number of recorded scores among successful calibration trajectories. Shorter score sequences are extended to $L_{\max}$ by repeating their final score. Thresholds are then constructed as a function of the absolute score index.

This parameterization can be applied online using only the current timestep and does not require knowing the final duration of the test rollout. However, when rollout lengths vary, the same absolute timestep may correspond to different stages of task execution across trajectories. If a test rollout contains more than $L_{\max}$ recorded scores, the final calibrated threshold is used for all subsequent positions.

Normalized time. Normalized-time calibration aligns trajectories according to relative execution progress rather than absolute timestep. For calibration rollout i, we define normalized progress as

$$u_{i,t} = \frac{t}{T_i - 1},$$

and linearly interpolate its risk trajectory onto a common grid of G normalized positions between 0 and 1. We use $G = 100$ in all experiments. This representation removes differences in raw rollout duration during calibration and aligns trajectories according to their relative progress.

At deployment, the final duration of an ongoing rollout is unavailable. We therefore define a reference rollout length

$$T_{\text{ref}} = \frac{1}{N} \sum_{i=1}^{N} T_i$$

using only successful calibration rollouts. The current timestep is mapped to normalized progress using T_{ref} and then to the corresponding position on the calibrated grid. If execution exceeds T_{ref}, the threshold at the final normalized position is retained. In this way, no information about the final duration of the test rollout is used.

Staged time. Staged-time calibration provides a coarser representation of relative execution progress. Each successful calibration trajectory is divided into K equally spaced stages according to normalized time, and all risk scores within the same stage share one threshold. We use $K = 10$ in all experiments. This produces a piecewise-constant threshold trajectory while preserving the coarse progression of risk over an execution.

For pointwise calibration, successful scores assigned to the same stage are pooled when estimating that stage's upper quantile. For functional calibration, we instead take the maximum score within each stage of each successful rollout. This yields one K-dimensional stage-level trajectory per rollout, to which the functional-band procedure is applied. At deployment, the current timestep is mapped to a stage using the same reference rollout length T_{ref}. If execution extends beyond T_{ref}, the threshold of the final stage is retained.

E.2 Pointwise Empirical Quantile Calibration

For normalized-time and absolute-time parameterizations, pointwise calibration estimates an upper empirical quantile independently at each aligned temporal position. Let $\widetilde{s}_{i,j}$ denote the aligned score of calibration rollout i at temporal position j. The threshold is

$$\delta_j(\alpha) = Q_{1-\alpha}\left(\widetilde{s}_{1,j}, \ldots, \widetilde{s}_{N,j}\right), \tag{6}$$

where $Q_{1-\alpha}$ denotes the empirical $(1-\alpha)$-quantile. For the staged-time parameterization, the same construction is applied to all successful scores assigned to each stage.

Pointwise calibration follows local changes in the nominal score distribution, but different temporal positions are calibrated independently. A pointwise significance level therefore does not imply that an entire successful trajectory remains below the threshold simultaneously. We use empirical upper quantiles directly and interpret α as the significance level indexing the resulting operating point.

E.3 Functional Conformal Calibration

Conformal prediction provides a distribution-free framework for calibration under exchangeability (Vovk et al., 2005; Angelopoulos & Bates, 2023). Functional conformal prediction extends this construction to function-valued observations and simultaneous prediction bands (Diquigiovanni et al., 2022; 2025). Trajectory-level calibration has also been adopted in recent runtime failure monitoring methods (Xu et al., 2025; Gu et al., 2025; Park et al., 2026; Zhang et al., 2026).

In our setting, an entire successful risk trajectory is treated as one functional observation. For each temporal parameterization, the calibration set is divided into two disjoint subsets, $\mathcal{D}_A$ and $\mathcal{D}_B$. The first estimates the nominal trajectory and its temporal variation, while the second calibrates the width of the upper band. In practice, approximately 30% of successful trajectories estimate the nominal trajectory and modulation function, while approximately 70% calibrate the final band width.

For normalized-time and absolute-time representations, let $\widetilde{s}_{i,j}$ denote the aligned score at temporal position j. We compute the mean successful trajectory from $\mathcal{D}_A$ as $\mu_j = |\mathcal{D}_A|^{-1} \sum_{\mathbf{s}_i \in \mathcal{D}_A} \widetilde{s}_{i,j}$.

Successful trajectories can exhibit different variability across execution. We therefore use a temporal modulation function m_j. For each trajectory in $\mathcal{D}_A$, we compute its maximum absolute deviation $d_i = \max_j |\widetilde{s}_{i,j} - \mu_j|$. To estimate the modulation function, let $n_A = |\mathcal{D}_A|$ and $k_\alpha = \lceil (n_A + 1)(1-\alpha) \rceil$. If $k_\alpha \le n_A$, let γ_α be the k_α-th smallest value among the deviations $\{d_i\}$ and retain $\mathcal{H} = \{\mathbf{s}_i \in \mathcal{D}_A : d_i \le \gamma_\alpha\}$. Otherwise, we retain all trajectories, setting $\mathcal{H} = \mathcal{D}_A$. The modulation function is then $m_j = \max_{\mathbf{s}_i \in \mathcal{H}} |\widetilde{s}_{i,j} - \mu_j| + \epsilon$.

For each trajectory in $\mathcal{D}_B$, the one-sided functional nonconformity score is

$$R_i = \max_j \frac{\widetilde{s}_{i,j} - \mu_j}{m_j}. \tag{7}$$

Let $q_{1-\alpha}$ denote the empirical $(1-\alpha)$-quantile of these nonconformity scores. The resulting upper band is

$$\delta_j(\alpha) = \mu_j + q_{1-\alpha} m_j. \tag{8}$$

The coefficient $q_{1-\alpha}$ is shared across the temporal domain, while m_j allows the band width to vary with the nominal score variability.

The same construction is applied to the staged-time parameterization. In this case, the center and modulation are indexed by stage, yielding piecewise-constant values μ_k and m_k. For trajectory i, the nonconformity score is computed by taking the maximum standardized upward deviation across all timesteps using the corresponding stage statistics, and the threshold for stage k is $\delta_k(\alpha) = \mu_k + q_{1-\alpha} m_k$.

We estimate the final band coefficient using empirical quantiles. The resulting band targets a coverage level of $1-\alpha$ for successful trajectories, without a finite-sample coverage guarantee. We assess its actual false-alarm behavior on the held-out evaluation split.

E.4 Independent Calibration and OR Fusion

The significance level α controls the sensitivity of each calibrated monitoring branch. A smaller α generally produces a higher and more conservative threshold, which tends to reduce false alarms but may delay or miss failures. A larger α lowers the threshold and increases sensitivity, typically yielding earlier alarms at the cost of more false positives.

Previous calibration-based monitors typically associate α with the monitoring score that directly determines the final alarm (Gu et al., 2025; Xu et al., 2025; Park et al., 2026; Zhang et al., 2026). For example, SAFE (Gu et al., 2025) uses functional conformal prediction to bound the probability of raising any alarm during a successful rollout by α, under the exchangeability assumption. In ReAP, the detector and predictor are calibrated separately at the same α, and their alarms are combined using an OR rule. A branch-level guarantee therefore does not directly imply the same guarantee for the combined monitor.

Let A_{det} and A_{pred} denote the events that the respective branches raise an alarm on a successful rollout. If each branch satisfies $\Pr(A_{\text{det}}) \leq \alpha$ and $\Pr(A_{\text{pred}}) \leq \alpha$, the union bound gives

$$\Pr(A_{\text{det}} \cup A_{\text{pred}}) \leq \Pr(A_{\text{det}}) + \Pr(A_{\text{pred}}) \leq 2\alpha.$$

This bound does not require independence between the two branches. It describes how branch-level false-alarm guarantees would translate under OR fusion. In our implementation, α serves as a nominal branch-level calibration parameter, and the false-alarm rate of the combined monitor is evaluated empirically on the held-out evaluation split.

F Rollout Visualization

We visualize five representative OpenVLA rollouts on LIBERO-10 in Figures 9–13, including three failed and two successful executions. The first two failures involve a mug tilting during placement and a gripper deviating from the intended trajectory while approaching the target object. In both cases, the predictor raises an alarm first, followed by the detector as execution continues and the failure becomes more visually apparent. In the third failure, the gripper rotates excessively during grasping. The detector remains below its threshold, while the action-conditioned predictor raises an alarm for such excessive rotation. Neither component raises an alarm in the two successful rollouts. These examples illustrate how the detector and predictor can provide complementary monitoring signals during execution.

We additionally visualize rollouts from RoboTwin-8 (Figures 14–17) and the three real-world tasks, Stack Plates (Figures 18–21), Pick Cubes (Figures 22–25), and Fold Cloth (Figures 26–28). Within each setting, all visualized rollouts use the same calibration variant and significance level α. Detector and predictor thresholds are calibrated separately and then held fixed across the visualized rollouts in that setting.

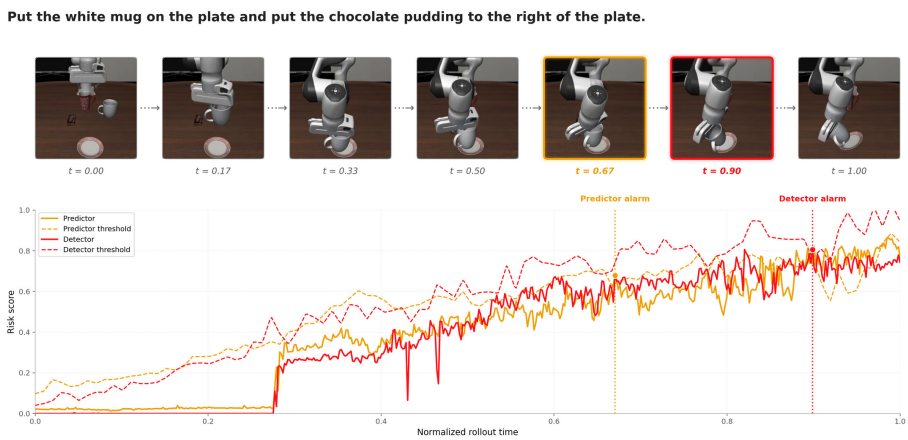


Figure 9: Qualitative monitoring example on LIBERO-10 with OpenVLA. The mug tilts during placement, leaving the task incomplete. The predictor raises an alarm first, followed by the detector.

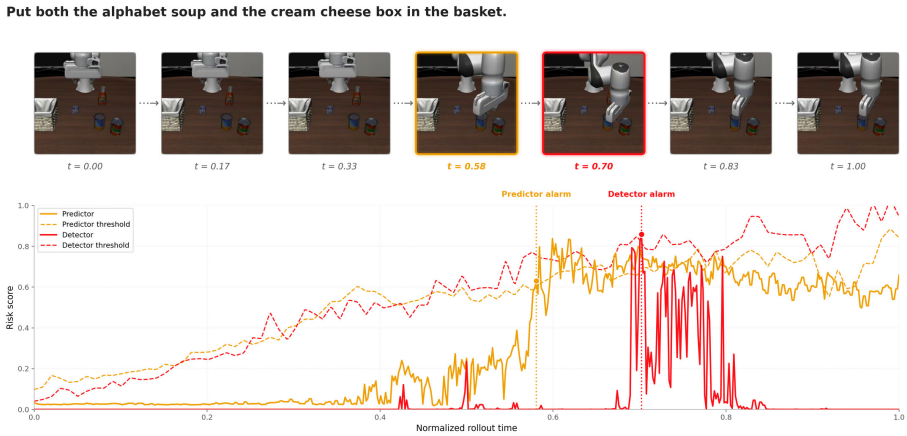


Figure 10: Qualitative monitoring example on LIBERO-10 with OpenVLA. The gripper deviates while approaching the target objects. The predictor raises an alarm first, followed by the detector.

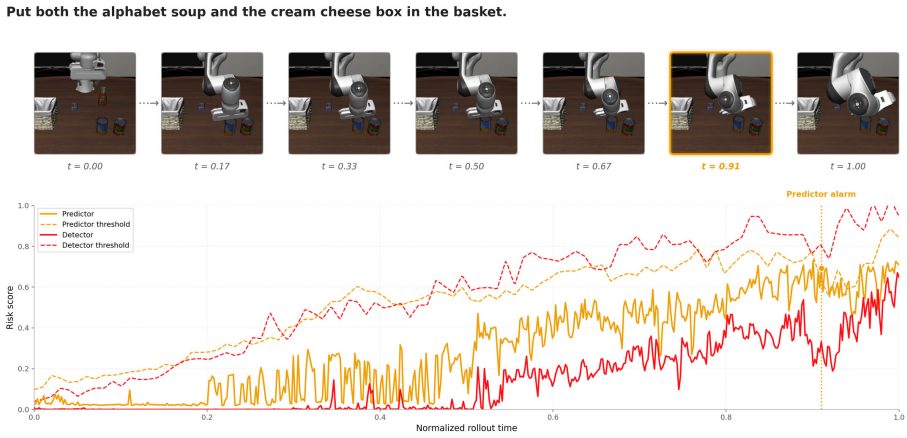


Figure 11: Qualitative monitoring example on LIBERO-10 with OpenVLA. The gripper rotates excessively during grasping. Only the predictor raises an alarm.

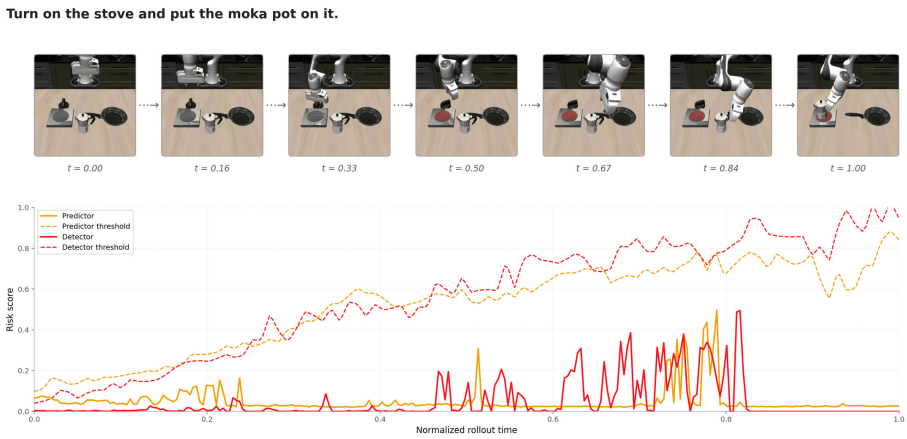


Figure 12: Qualitative monitoring example on LIBERO-10 with OpenVLA. The robot successfully turns on the stove and places the moka pot on it.

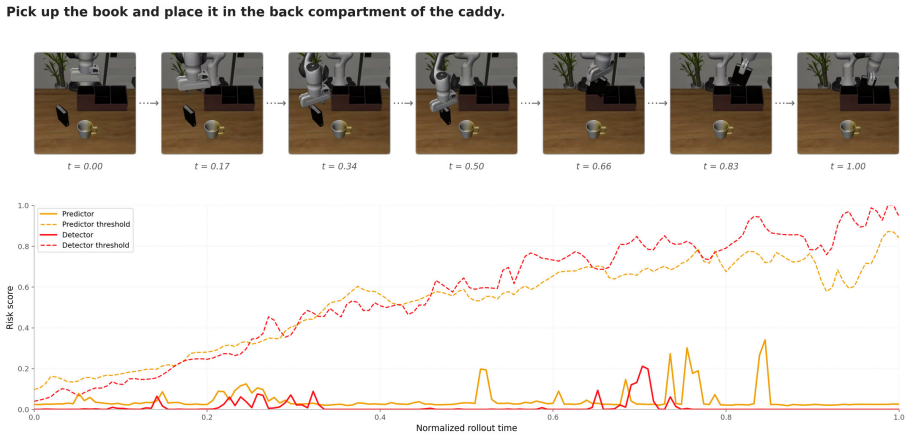


Figure 13: Qualitative monitoring example on LIBERO-10 with OpenVLA. The robot successfully picks up the book and places it in the back compartment of the caddy.

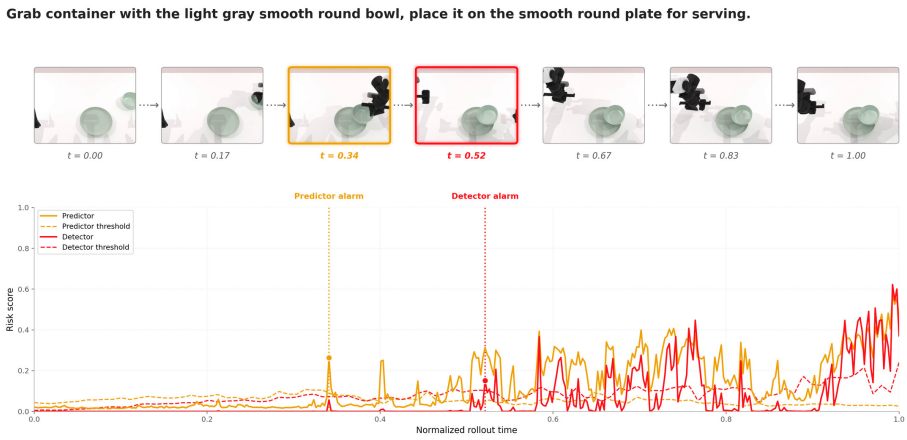


Figure 14: Qualitative monitoring example on RoboTwin-8 with $\pi_{0.5}$. The bowl is placed off-center at the edge of the plate. The predictor raises an alarm first, followed by the detector.

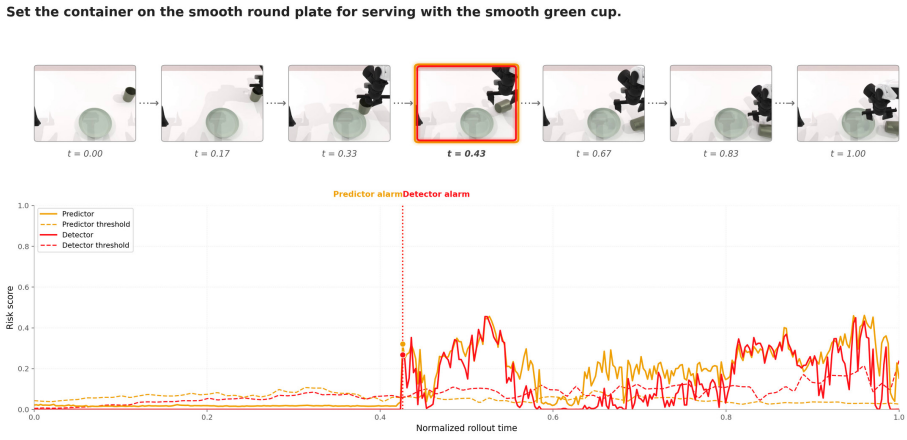


Figure 15: Qualitative monitoring example on RoboTwin-8 with $\pi_{0.5}$. The cup tips over beside the plate. The detector and predictor raise alarms at nearly the same time.



Figure 16: Qualitative monitoring example on RoboTwin-8 with $\pi_{0.5}$. The gripper remains near the clock without visibly completing the button press. Only the predictor raises an alarm.

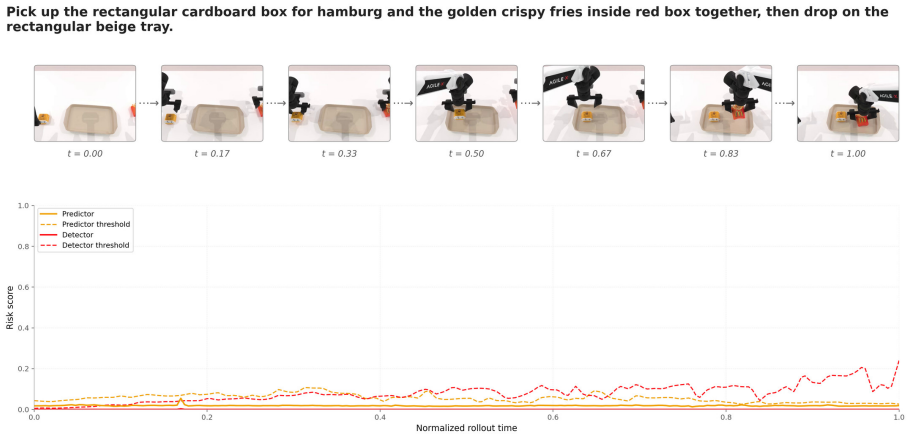


Figure 17: Qualitative monitoring example on RoboTwin-8 with $\pi_{0.5}$. The robot successfully transfers the hamburger box and fries container onto the tray.

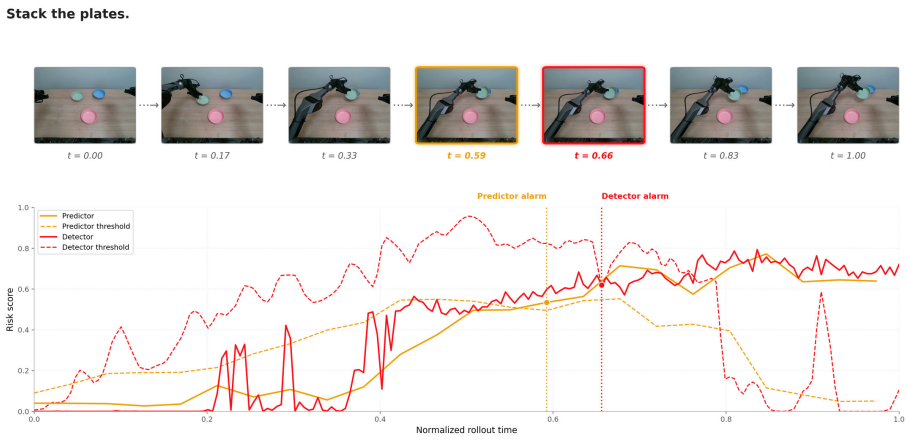


Figure 18: Qualitative monitoring example on Stack Plates with $\pi_{0.5}$. The robot repeatedly attempts to manipulate the plates. The predictor raises an alarm first, followed by the detector.

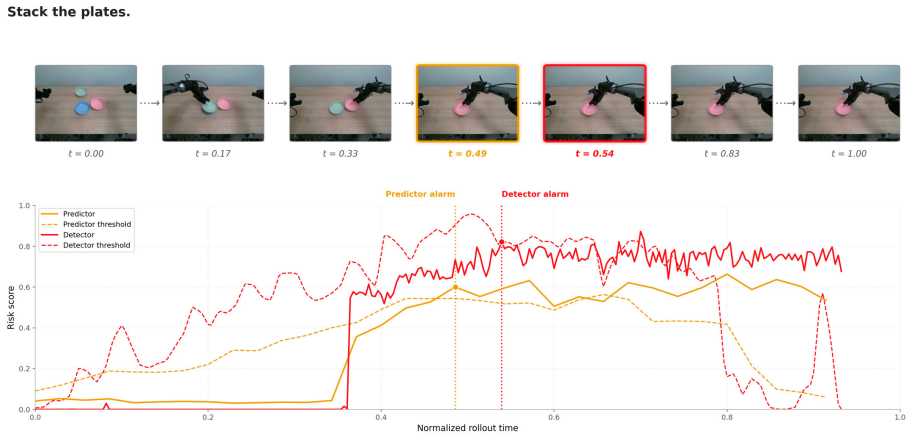


Figure 19: Qualitative monitoring example on Stack Plates with $\pi_{0.5}$. The robot stalls while attempting to grasp the plate. The predictor raises an alarm first, followed by the detector.

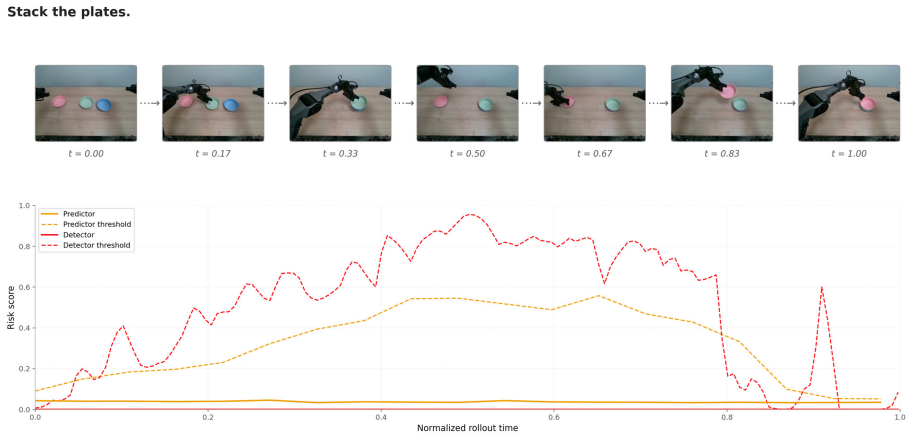


Figure 20: Qualitative monitoring example on Stack Plates with $\pi_{0.5}$. The robot successfully stacks the plates together.

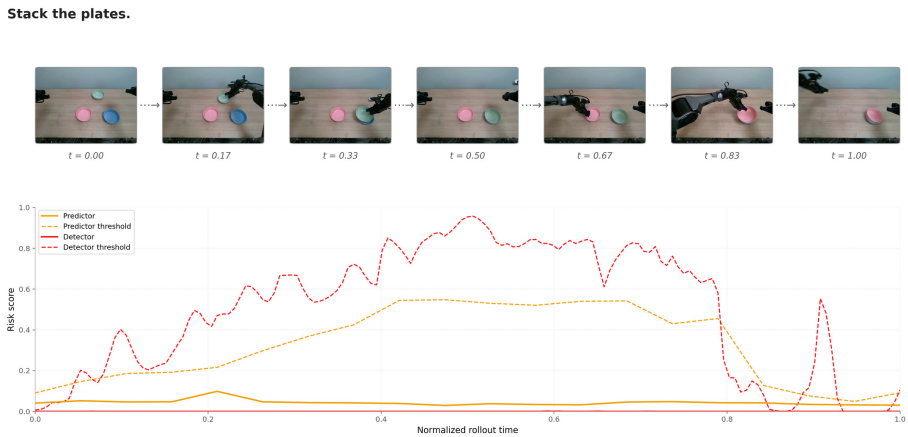


Figure 21: Qualitative monitoring example on Stack Plates with $\pi_{0.5}$. The robot successfully stacks the plates together.



Figure 22: Qualitative monitoring example on Pick Cubes with $\pi_{0.5}$. The robot leaves a cube on the table after further grasp attempts. The predictor raises an alarm first, followed by the detector.

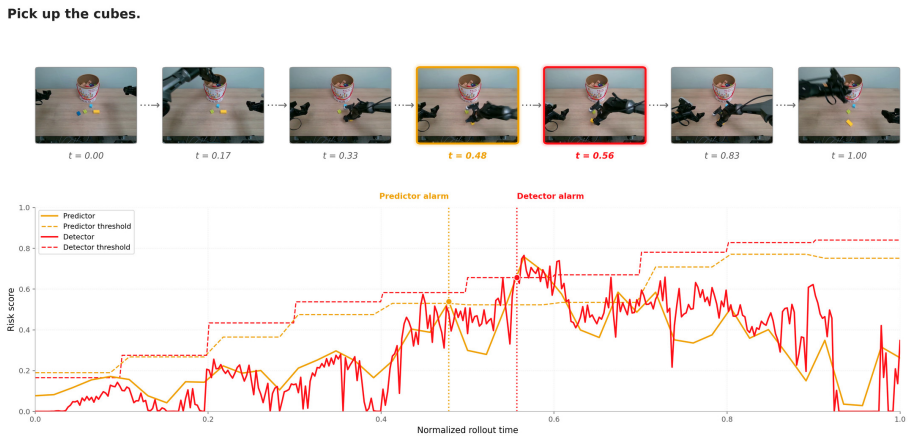


Figure 23: Qualitative monitoring example on Pick Cubes with $\pi_{0.5}$. Repeated grasp attempts leave cubes outside the container. The predictor raises an alarm first, followed by the detector.

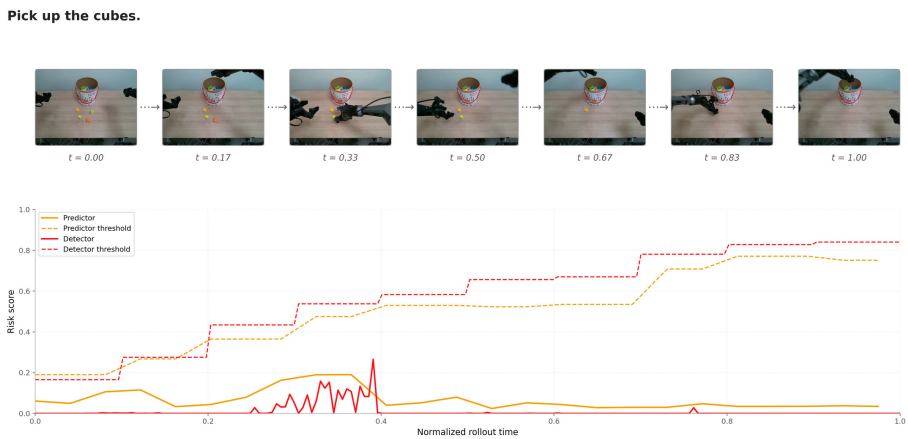


Figure 24: Qualitative monitoring example on Pick Cubes with $\pi_{0.5}$. The robot successfully picks up the cubes and transfers them into the container.

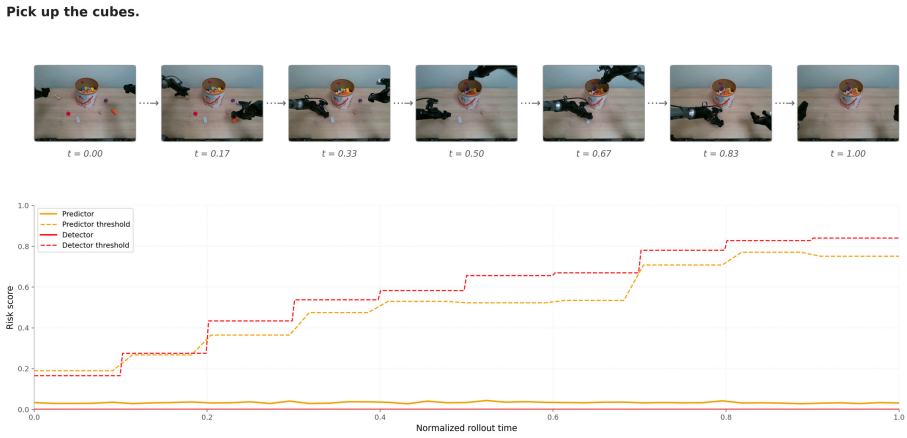


Figure 25: Qualitative monitoring example on Pick Cubes with $\pi_{0.5}$. The robot successfully picks up the cubes and transfers them into the container.

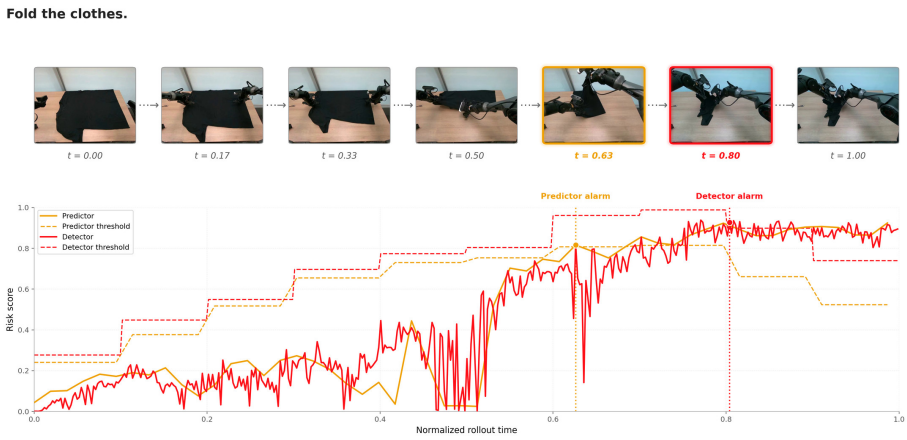


Figure 26: Qualitative monitoring example on Fold Cloth with $\pi_{0.5}$. The cloth slips from the right gripper. The predictor raises an alarm first, followed by the detector.

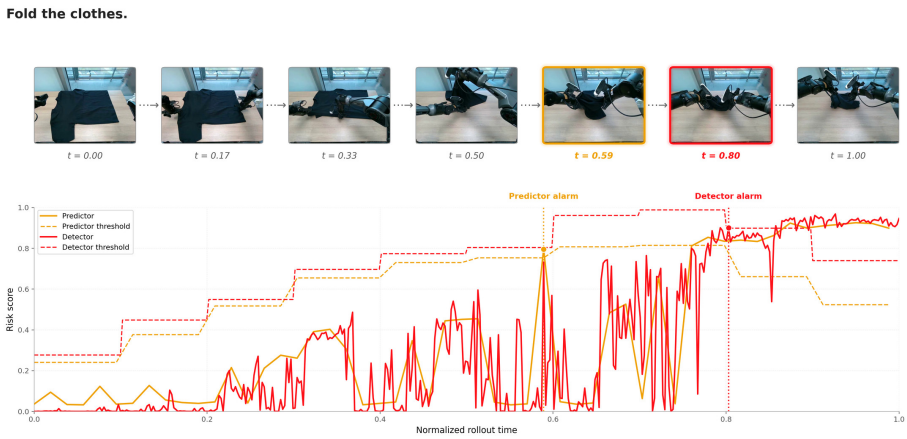


Figure 27: Qualitative monitoring example on Fold Cloth with $\pi_{0.5}$. The cloth bunches between the grippers. The predictor raises an alarm first, followed by the detector.

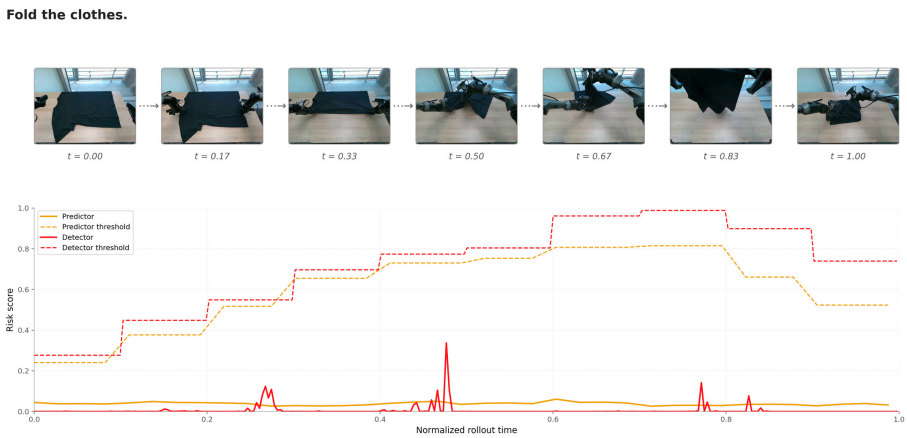


Figure 28: Qualitative monitoring example on Fold Cloth with $\pi_{0.5}$. The robot successfully folds the cloth through coordinated lifting and placement.

G Representation Visualization

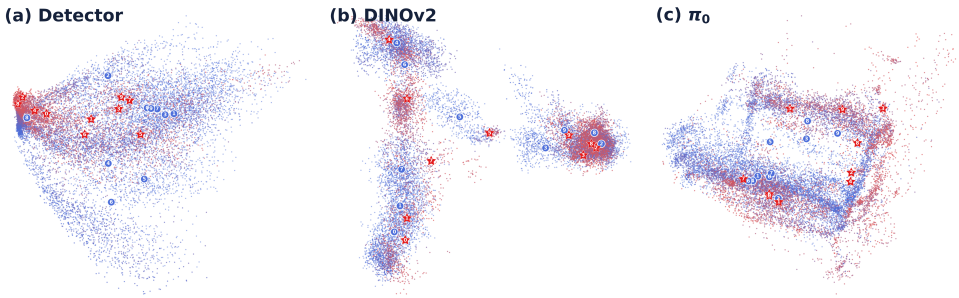


Figure 29: PCA visualization of representations on π_0/LIBERO-10.

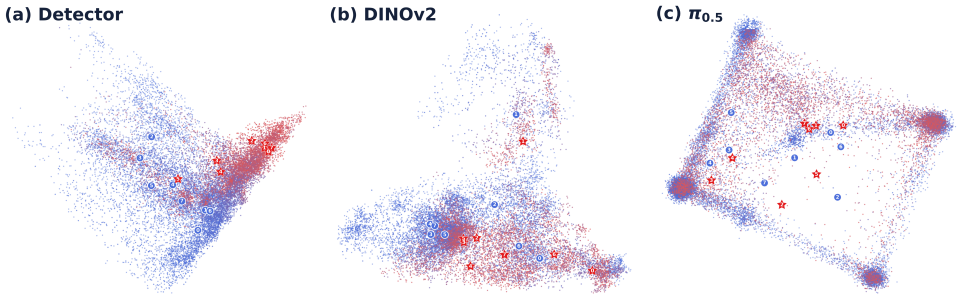


Figure 30: PCA visualization of representations on $\pi_{0.5}$/RoboTwin-8.

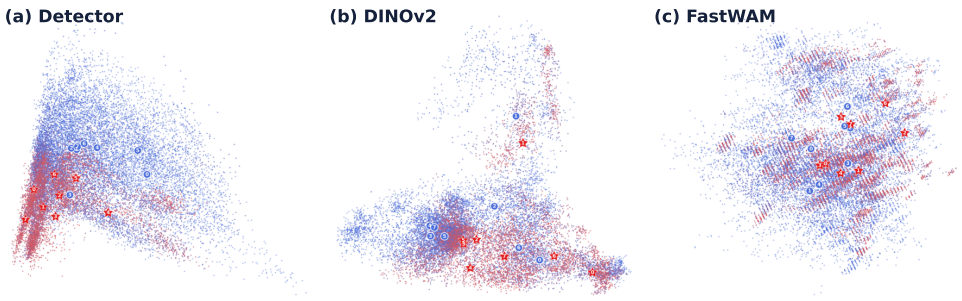


Figure 31: PCA visualization of representations on Fast-WAM/RoboTwin-8.