

Progress Probes P^2 : Lightweight Runtime Monitoring for Vision-Language-Action Models

Atiksh Bhardwaj^{1,2}, Edward W. Duan², Junwon Seo¹, Prithwish Dan²,
Wei-Chiu Ma², Andrea Bajcsy^{1,†}, Preston Culbertson^{2,†}

Abstract—Vision-Language-Action (VLA) models can perform diverse manipulation tasks directly from visual observations and natural language instructions, but they remain prone to mistakes and unpredictable failures. Deploying these policies without continuous human supervision will require robust runtime monitors that detect when the policy has gone off-course. Existing monitors use external foundation models, such as vision-language models (VLMs) to supervise the policy, introducing significant complexity and computational cost for monitoring at runtime. In this work, we investigate an alternate approach: *can a VLA’s own internal activations provide the information needed to monitor its execution?* Our method trains linear probes on policy activations from demonstrations to estimate progress, then flags executions that deviate from expected progress. In experiments with the open-source VLA $\pi_{0.5}$ and across simulated and real-world manipulation tasks, this approach is competitive with or outperforms VLM-based monitors in task failure detection, as well as methods which require failure data for training. These findings suggest that simple probes of VLA internals can be used to develop reliable runtime monitors, opening a path toward building policies that can act reliably and monitor their execution without external foundation models.

I. INTRODUCTION

Vision-Language-Action models (VLAs) have made strong progress toward the goal of generalist, language-conditioned robot policies. Models such as $\pi_{0.5}$ [1] and GROOT [2] can execute diverse, multi-stage manipulation tasks across a variety of tasks and embodiment and, especially after task-specific fine-tuning, can achieve relatively high multi-task success rates. However, even well-trained policies still experience a wide variety of failure modes, (e.g., poor language grounding [3], deadlocks, and mistakes [4]) which are often sudden and difficult to predict.

A natural framework for addressing such failure modes during deployment is *runtime monitoring*: using an external verifier or monitor to detect when the policy’s execution is straying from typical/successful deployment. Existing approaches obtain this monitoring signal from several sources. Sentinel [5] uses an external VLM, along with an action consistency heuristic, to judge when the policy is failing to make progress on a particular task. Generalist reward models, such as Robometer [6], learn to judge task progress from datasets of robot videos. Other methods instead use policy internals directly for failure detection. For example, SAFE [4] trains a failure classifier to predict rollout success directly from policy activations, while FIPER [7] combines

an internal-based anomaly score and an uncertainty metric over actions to flag anomalous rollouts.

In this work, we investigate a different use of policy internals for runtime monitoring. Specifically, we build on recent successes in *mechanistic interpretability* for VLAs [8, 9], which demonstrated that the activations of the VLM backbone contain easily-decodable representations of task-relevant features such as end-effector pose, object states, and task phase. We hypothesize that a VLA’s internal activations may also encode *task progress*: how far the policy has progressed through a task. In successful executions, progress advances predictably over time (monotonically decreasing), while in suboptimal or failed executions, the progress probe falls behind the elapsed time. Furthermore, task progress can be learned entirely from successful executions and can be labeled without human supervision.

We propose progress probes—linear decodings from the VLA’s activations to task progress—as lightweight internal-based runtime monitors.

We study progress probes by training them on the $\pi_{0.5}$ [1] VLA using a dataset of success-only task executions. At deployment, runtime monitoring consists of taking the VLA’s internal activations (that are already computed as part of the forward pass of inference), passing them through the linear progress probe, and measuring the deviation between the current estimated task progress and the expected task progress computed from successful demonstrations. Not only does our probe-based monitor require only successful data for training, but it also runs between $5\times$ and $20,000\times$ faster than an external foundation-model monitor.

Across simulated and real-world manipulation tasks, we find that linear probes can indeed accurately predict task progress for demonstrations, and that they provide an effective signal for detecting unsuccessful policy rollouts. Our monitor achieves competitive or superior failure detection performance with both Robometer and SAFE baselines, while providing fast inference and requiring only successful demonstrations for training. This result indicates that even when VLAs fail to generate successful actions, their activations still retain information useful for detecting task progress and failure, which can be obtained using a simple linear readout. Specifically, our contributions are as follows:

- 1) We demonstrate empirically that linear probes trained over demonstration data can accurately decode task progress from $\pi'_{0.5}$ s vision-language representations,

¹Carnegie Mellon University. ²Cornell University. [†]Equal advising.

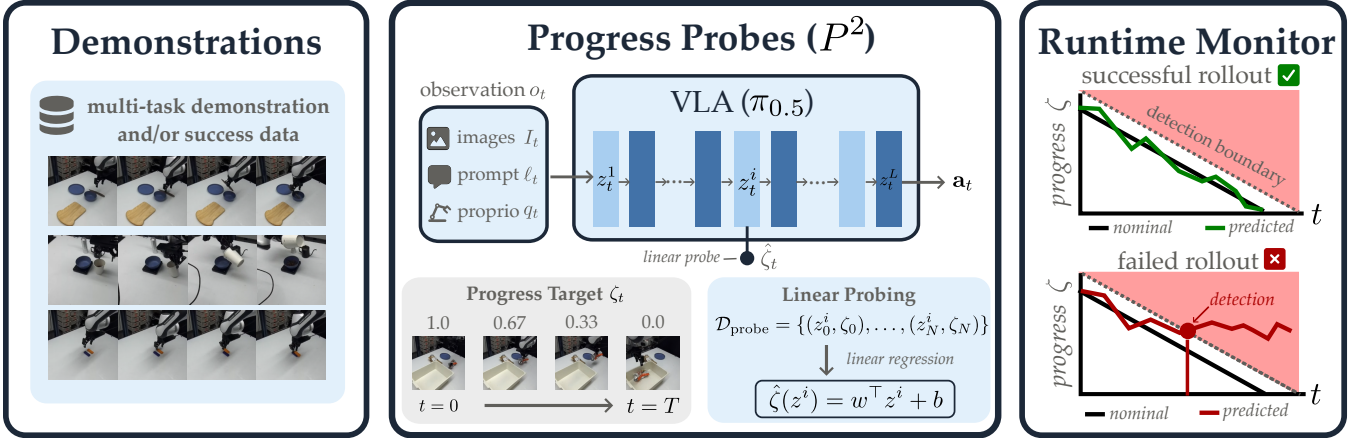


Fig. 1: Progress Probes (P^2) decode task progress directly from the VLA’s internal activations. Using only successful demonstrations, we assign each timestep a normalized progress ζ_t and fit a linear probe to predict it from z_t^i , the activation at layer i . At runtime, successful executions follow the expected progression through the task, whereas failed executions cause the decoded progress to lag or deviate from this nominal trajectory. Thus, the policy’s own representations enable us to construct a lightweight runtime monitor without failure data or an external foundation model.

- across simulated and real-world manipulation tasks.
- 2) We develop a minimalist progress-based runtime monitor which uses these probes to detect deviations from expected progress at runtime, without failure data or an external VLM.
- 3) We evaluate our monitor on 6 VLABench tasks and 6 real-world manipulation tasks, achieving AUROC scores of 0.788 and 0.78 respectively, matching or exceeding competitive baselines including SAFE and Robometer.

II. RELATED WORK

Interpretability of Robot Policies. Mechanistic interpretability seeks to understand the behavior of neural networks by studying their internal representations and computations. Over the past decade, researchers in computer vision and language modeling have developed tools to understand what neurons learn and how semantic concepts and structured knowledge are encoded and used. One particularly exciting perspective is the *linear representation hypothesis* [10, 11], which posits that high-level concepts are encoded as directions in activation space and that interventions along concept directions can steer model behavior [12]. This allows one to connect the study of representations to their functional role and reveal behaviors not visible in model output [13, 14].

More recently, researchers have extended similar ideas to robotic VLAs, investigating how their internal representations encode semantic concepts [15], motion primitives [16], and physical and relational states [8, 9]. Our work contributes to this growing body of interpretability research on robotic VLAs by investigating whether linear probes trained only on successful demonstrations can extract a *task progress* signal from internal activations. Beyond decoding, we further show that this signal can be turned into a lightweight runtime monitor that detects departures from expected progress,

demonstrating how VLA internals can support failure detection without an external foundation model.

Runtime Monitoring of Robot Foundation Models. Learned robot policies, including VLAs, remain prone to unexpected failures during deployment, motivating techniques for monitoring policies at runtime and detecting failures as they occur. Existing runtime monitors differ primarily in the signals they use to recognize such failures. One approach is to use deviations from successful behavior, flagging observations or policy outputs that appear anomalous relative to successful executions [7, 17]. Another is to leverage an external evaluator, such as a VLM judge [5, 18, 19] or a learned failure predictor [4] trained directly over policy internals. More general reward models can also serve as runtime monitors by recognizing abnormal task progression [6]. Overall, several design decisions remain open for runtime monitoring, including whether monitoring requires an external validator, whether it must be trained on failure data, and whether the policy’s own representations provide useful signals for monitoring. We study the latter hypothesis by probing these representations for task progress and detecting failures when estimated progress deviates from the progression expected during successful execution.

III. BACKGROUND: MECHANISTIC INTERPRETABILITY VIA LINEAR PROBING

We focus on a class of mechanistic interpretability techniques called *probing* [20] and provide a brief overview.

Setup. Let the VLA policy be denoted by $\pi(\mathbf{a}_t | o)$ where $\mathbf{a}_t = (a_t, \dots, a_{t+h}) \in \mathcal{A}^h$ is an h -step action chunk and the input observation $o_t = (I_t, q_t, \ell_t)$ consists of an image $I \in \mathcal{I}$, proprioceptive data $q \in \mathcal{Q}$, and a language prompt $\ell \in \mathcal{L}$. Many VLA policies, including the one we study $\pi_{0.5}$ [1], are built on top of vision-language backbones (here, PaliGemma [21]) and use the transformer architecture; thus, we focus on this model class throughout the paper.

Activation Extraction & Probe Dataset Construction.

The input observation sequence o is first tokenized and then processed via a sequence of L layers of transformer blocks in π , producing a sequence of intermediate activations at each layer. Let the activation for any layer i be denoted by $z^i \in \mathbb{R}^d$. Each activation is computed sequentially via a residual function $f^i: \mathbb{R}^d \rightarrow \mathbb{R}^d$,

$$z^{i+1} = z^i + f^i(z^i), \quad i \in \{0, \dots, L-1\}.$$

Given a dataset of N robot trajectories, $\mathcal{D} = \{(o_t, a_t)\}_{n=1}^N$ where $t = 0, \dots, T$, we extract¹ the activations from the VLA of interest by feeding in each observation o_t and computing the intermediate layer activations $(z_t^0, z_t^1, \dots, z_t^L)$. Let ζ be the target feature we seek to linearly “probe out” of the VLA. For each trajectory and timestep in $\mathcal{D}$, we label the target feature. Thus, we finally create our probe training dataset which consists of internal activations and the target feature labels for each of the N trajectories:

$$\mathcal{D}_{\text{probe}} = \{(z_t^{0,n}, z_t^{1,n}, \dots, z_t^{L,n}, \zeta_t^n)\}_{n=1}^N.$$

Linear Probing. Our goal is to fit a linear function on the extracted activations. Without loss of generality, consider a layer i and a scalar target feature $\zeta \in \mathbb{R}$. Our *linear probe* [22] is a map from the activations to the target feature:

$$\phi(z^i) = w^\top z^i + b,$$

with parameters w, b , trained on the train data split in $\mathcal{D}_{\text{probe}}$.

IV. PROBLEM FORMULATION: RUNTIME MONITORING

Our goal is to detect, during execution, when a VLA policy $\pi(a_t | o_t)$ is unlikely to complete its task. We follow the finite-horizon runtime monitoring setup of Sentinel [5].

Setup. We observe a stream of robot observations and actions up until any time $t \geq 0$: let $\tau_t = (o_0, a_0, \dots, o_t)$ be this observation-action trajectory. Let H denote the maximum task horizon. Finally, we define the rollout outcome $Y \in \{0, 1\}$, where $Y = 1$ if the policy reaches a successful state within H steps and $Y = 0$ otherwise.

Monitor. A *runtime monitor*, $s(\tau_t) \in \mathbb{R}$, is a map from the observation-action trajectory to a scalar value measuring how much the data differs from successful or “nominal” data (i.e. larger values imply more different from successful). Given a threshold $\delta \in \mathbb{R}$ and a trajectory τ , let the *detection time* (i.e., the time at which the monitor detects deviation from success) be $T_\delta = \inf\{t \mid s(\tau_t) > \delta\}$, with $T_\delta = \infty$ if the monitor never fires.

Monitor Design Objective. The objective of runtime monitoring is to design a $s(\tau)$ which (a) reliably discriminates successful trajectories from failures and (b) detects failures early. Formally, this is a multi-objective design problem:

$$\min_s \mathbb{E}_\tau [\text{FPR}(\delta, \tau; s) + \lambda \cdot \text{DetectTime}(\delta, \tau; s)], \quad (1)$$

¹Each activation z_t^i has shape $d \times k$, where d is the size of the activation and k is the number of tokens; we mean-pool across the tokens to get d .

for a given threshold δ , where $\lambda > 0$ is a parameter balancing the two objectives. Here, the first term is the false positive rate of the monitor, $\text{FPR}(\delta, \tau; s) = P(T_\delta \leq H \mid Y = 1)$. The second term is the mean detection time on failed trajectories, $\text{DetectTime}(\delta, \tau; s) = \mathbb{E}[T_\delta \mid Y = 0, T_\delta \leq H]$. An ideal runtime monitor minimizes false positives while also firing early under the true trajectory distribution (i.e., τ should ideally be drawn from the distribution of closed-loop evaluation trajectories).

Goal. We hypothesize that we can design $s(\tau)$ to be a linear probe (described in Sect. III) trained on the VLA’s activations, achieving monitoring quality while being lightweight to run at inference time.

V. PROGRESS PROBES AS RUNTIME MONITORS

In this work, we propose using *progress probes* (P^2), lightweight linear readouts of task progress from a VLA’s internal activations, as runtime monitors. Our key insight is that a VLA’s internal activations encode how far the policy has progressed through a task, and in successful executions this encoded progress advances predictably (monotonically) with elapsed time. Suboptimal or failed behaviors violate this relationship, and thus the progress probe falls behind the elapsed time. Practically, task progress is also a useful signal for designing a runtime monitor because it can be easily labeled without human supervision from offline data (because it decreases monotonically on successful data) and can be learned without explicit failure data.

Progress Probes. Following Sec. III, we train linear probes to predict normalized task progress from the policy’s internal activations, given a dataset of successful trajectories. Let $\tau = (o_0, a_0, \dots, o_T)$ denote a successful trajectory of length $T \leq H$. For each observation $o_t \in \tau$, we label its task progress using the normalized time remaining in the trajectory:

$$\zeta_t = 1 - \frac{t}{T}, \quad (2)$$

such that ζ_t decreases from 1 at the start to 0 at the end.

We construct our probing dataset $\mathcal{D}_{\text{probe}}$ by passing all N successful demonstrations through the VLA and extracting the model’s activations (as in Sec. III):

$$\mathcal{D}_{\text{probe}} = \{(z_t^{0,n}, z_t^{1,n}, \dots, z_t^{L,n}, \zeta_t^n)\}_{n=1, \dots, N; t=0, \dots, T^n}. \quad (3)$$

Our probe is trained via linear regression to approximately predict progress via that layer’s activations,

$$\hat{\zeta}(o) = \phi(z^i),$$

where z^i is the activation of the i th layer when observation o is input to the VLA model.

Runtime Monitoring via Progress Probes. We convert our progress probe into a runtime monitor by comparing its prediction to the expected progress at the current timestep. Let μ_T denote the mean successful trajectory length: $\mu_T = \frac{1}{N} \sum_{n=1}^N T^n$. During *nominal, successful* execution of the policy, the expected progress at time t is simply $\zeta_t^{\text{nom}} =$

$1 - \frac{t}{\mu_T}$. Thus, our monitor score from Sec. IV is simply the deviation from the expected nominal task progress time:

$$s(\tau_t) = \hat{\zeta}(o_t) - \zeta_t^{\text{nom}}. \quad (4)$$

Since larger progress values correspond to earlier stages of the task, a positive score indicates that the policy’s estimated progress lags behind the nominal trajectory; a negative score indicates that it is ahead. We do not penalize faster execution. Instead, we predict failures when execution falls sufficiently behind its nominal progression, i.e., when $s(\tau_t) > \delta$.

We note that this runtime monitoring recipe is simple by construction. The probe can be trained using only successful task executions and, because the probe is evaluated on the activations that are already computed as part of VLA inference, it requires minimal additional runtime computation.

VI. EXPERIMENTS: SIMULATION

We first conduct simulation studies to understand (a) how discriminative and efficient progress probes are relative to other runtime monitors, (b) how progress probes compare to VLAs as runtime monitors, and (c) how progress probes of VLA internals, whose features have been learned using robot actions, compare to probes over VLM features, which are trained only on images and text.

Setup. We study the $\pi_{0.5}$ [1] VLA, fine-tuned on a dataset of 5,000 trajectories (500 per manipulation task for 10 tasks). All training and evaluations are done in VLABench [23]. For runtime monitoring, we evaluate on 6 tasks for which the VLA achieves a non-negligible success rate.

Training. We construct the probe dataset $\mathcal{D}_{\text{probe}}$ by passing $N = 300$ trajectories (50 per task across the 6 tasks) through the VLA and recording its activations, as in Eq. 3. We use an 80/20 train-validation split of $\mathcal{D}_{\text{probe}}$ to train the probe, and use the 20% validation split to calibrate the detection threshold following [7].

Evaluation. We evaluate on 600 VLA rollout trajectories (100 per task), evenly split between successful and failed executions. For each failed rollout, we manually label the timestep at which the failure occurs and mark all subsequent timesteps as failures for frame-level evaluation.

Baselines. We compare Progress Probe (P^2) against prior runtime monitoring methods for VLAs, grouping them into three categories: **input**-level monitors that operate on visual observations, **feature**-level monitors that use internal VLA activations, and **output**-level monitors that operate on predicted actions. Table I summarizes the compared methods.

For **input**-level monitoring, we evaluate FIPER (RND-OE) [7, 24] and FAIL-Detect (logPO) [17]. We implement FAIL-Detect (logPO) using DINO image features rather than policy features to isolate monitoring based on the input observation. FIPER (RND-OE) corresponds to the observation-based component of FIPER and measures novelty in the visual input. For **feature**-level monitoring, we evaluate FAIL-Detect (logPZ0) [17], SAFE-MLP, SAFE-LSTM [4], and P^2 , all operating on the same VLA activations. For **output**-level

monitoring, we evaluate FIPER (ACE) and STAC [5], which monitor the predicted action distribution. All prior monitors are unsupervised except SAFE, which uses failure labels to learn and calibrate a failure score.

We additionally compare against Robometer [6], a VLM-based baseline trained on large-scale data containing both successful and failed trajectories to estimate task progress. This provides a direct comparison to P^2 , which also estimates task progress but learns the progress signal from successful trajectories alone. SAFE is trained on these 300 successful trajectories together with an additional 300 failed trajectories for supervision and calibration, while Robometer is evaluated both zero-shot and after fine-tuning on the same success-and-failure dataset used for SAFE.

A. On the Quality and Efficiency of Our Progress Probe

We first evaluate progress probe (and all baselines) on three capabilities: (a) efficiency, (b) capability to discriminate success and failure, and (c) generalizability to unseen tasks.

Setup. We break down our evaluation setup into:

- 1) **Efficiency by Inference Latency:** All methods are run on an NVIDIA RTX A6000 for inference latency.
- 2) **Discriminability via Failure Prediction:** We evaluate runtime failure prediction on the 600 VLA rollout trajectories at two levels: *Episode* and *State*. At the *Episode* level, we evaluate whether the monitor discriminates successful from failed trajectories by taking the maximum monitor score over each trajectory. At the *State* level, we evaluate whether the monitor discriminates successful from failed states using the score predicted at each timestep. We measure AUROC for failure prediction.
- 3) **Generalization to Unseen Tasks:** We evaluate how well failure discriminability generalizes to unseen tasks. For each split, we designate 3 of the 6 tasks as unseen and train the monitors on the remaining 3 seen tasks, evaluating all $\binom{6}{3} = 20$ possible splits. We exclude Robometer from this evaluation because it is primarily used zero-shot and is therefore not trained on the seen-task subset.
- 4) **Detection Time on Failures:** Following the objective in Eq. 1, a good monitor should minimize detection time, $\text{DetectTime}(\delta, \tau; s)$, while maintaining a low false-positive rate, $\text{FPR}(\delta, \tau; s)$. We therefore evaluate the *average detection time* on failed rollouts. We calibrate the detection thresholds for P^2 and Robometer following [7], while SAFE uses its prescribed conformal calibration procedure. At varying fractions of the episode, we report the true-positive rate (TPR) at a 5% false-positive rate (FPR). We measure detection time as the fraction of the episode elapsed when the monitor first detects a failure, averaged across failed rollouts.

Result #1: P^2 is Efficient Yet Effective. Fig. 2 and Table I show that **feature**-level monitors are generally more efficient, as they directly leverage VLA activations already computed during policy inference. Among these methods, P^2 achieves

Method	Supervision				Speed	All tasks		Unseen tasks	
	Input	Output	Act.	Labels	Latency (ms) ↓	Episode ↑	State ↑	Episode ↑	State ↑
FAIL-Detect (lopO) [17]	✓	×	×	×	7.3×10^0	0.881	0.857	0.667	0.656
FIPER (RND-OE) [7]	✓	×	×	×	2.5×10^{-1}	0.874	0.850	0.661	0.557
FIPER (ACE) [7]	×	✓	×	×	1.1×10^0	0.793	0.790	0.843	0.800
STAC [5]	×	✓	×	×	4.0×10^{-1}	0.477	0.442	0.482	0.434
SAFE-MLP [4]	×	×	✓	✓	1.2×10^{-1}	0.930	0.894	0.896	0.866
SAFE-LSTM [4]	×	×	✓	✓	1.8×10^{-1}	0.923	0.851	0.859	0.803
FAIL-Detect (logpZO) [17]	×	×	✓	×	1.7×10^{-1}	0.904	0.888	0.634	0.571
Robometer (progress) [6]	✓	×	×	✓	5.5×10^2	0.779	0.666	—	—
Robometer (success) [6]	✓	×	×	✓	5.5×10^2	0.514	0.316	—	—
Robometer FT (progress)	✓	×	×	✓	5.5×10^2	0.922	0.830	—	—
Robometer FT (success)	✓	×	×	✓	5.5×10^2	0.505	0.407	—	—
Progress Probe (P^2)	×	×	✓	×	2.6×10^{-2}	0.909	0.788	0.900	0.794

TABLE I: **Simulation: Runtime Failure Monitors.** Input, Output, Act., and Labels indicate whether a method uses image-level inputs, output action distributions, while the y-axis shows inference speed. internal VLA activations, and additional failure labels for training the runtime monitor, respectively. Latency is measured on an NVIDIA RTX A6000 GPU; Progress Probe achieves a favorable balance of high AUROC and fast inference without relying on failure labels.

Detector	TPR@5%FPR by episode fraction ↑				Det. time ↓
	25%	50%	75%	100%	(frac. of ep.)
Progress Probe (P^2)	0.00	0.19	0.54	0.64	0.60
SAFE-MLP	0.00	0.08	0.41	0.77	0.75
SAFE-LSTM	0.11	0.37	0.59	0.72	0.53
Robometer (progress)	0.01	0.08	0.21	0.33	0.67
Robometer FT (progress)	0.00	0.34	0.62	0.72	0.56

TABLE II: **Simulation: Detection Time Results.**

the lowest inference latency of 2.6×10^{-2} while maintaining competitive monitoring performance. By using a simple linear probe over VLA activations, P^2 is substantially faster than SAFE-MLP (1.2×10^{-1}), SAFE-LSTM (1.8×10^{-1}), and FAIL-Detect (logpZO) (1.7×10^{-1}), while achieving comparable failure-discrimination performance.

Result #2: P^2 Achieves Competitive Failure Discriminability. Overall, **feature-level** monitors achieve the strongest runtime monitoring performance, suggesting that internal VLA activations contain informative signals for distinguishing successful from failed executions. Notably, P^2 achieves an Episode AUROC of 0.909 and a State AUROC of 0.788 without using failure labels for training, remaining competitive with supervised baselines such as SAFE [4], which are trained with additional failure data.

Result #3: P^2 Generalizes Competitively to Unseen Tasks. P^2 remains competitive on unseen tasks, achieving an Episode AUROC of 0.900 and a State AUROC of 0.794. This is comparable to SAFE-MLP (0.896 Episode AUROC, 0.866 State AUROC) and SAFE-LSTM (0.859 Episode AUROC, 0.803 State AUROC). FIPER (ACE) also achieves similar performance (0.843 Episode AUROC, 0.800 State AUROC), suggesting that both **feature-level** and **output-level** monitors can generalize effectively to new tasks. Overall, P^2 maintains competitive discriminability on unseen tasks while remaining more efficient than prior runtime monitors.

Result #4: P^2 Achieves Competitive Detection Time with High Accuracy. Table II shows that P^2 maintains

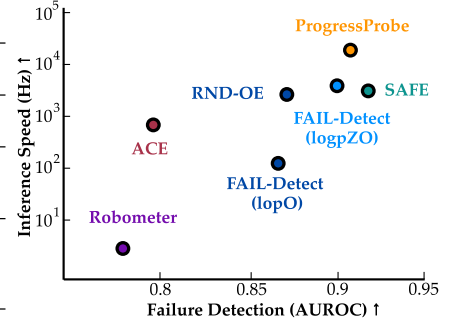


Fig. 2: **Accuracy vs. Speed.** The x-axis shows failure-detection AUROC, while the y-axis shows inference speed. Compared with prior runtime monitors, Progress Probe achieves a favorable balance of high AUROC and fast inference without relying on failure labels.

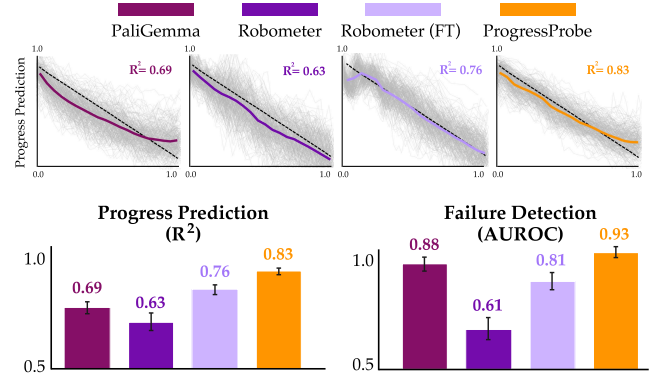


Fig. 3: **Simulation: Progress Probe vs. VLMs.** (Top): Progress prediction results using progress probes trained on different representations. (Bottom): Progress prediction mean- R^2 and failure detection results using progress probes trained on different representations.

competitive TPR throughout the episode, reaching 0.54 at 75% and 0.64 at 100% of the episode. On average, P^2 detects failures after 0.60 of the episode, comparable to SAFE-LSTM at 0.53. Notably, SAFE-LSTM explicitly models temporal history through its recurrent architecture, whereas P^2 uses only the current VLA activation with a simple linear probe. Despite this simplicity, P^2 achieves competitive detection timing, demonstrating that Progress Probe effectively captures the objective in Eq. 1 of detecting failures early while maintaining a low false-positive rate.

B. VLA Progress Probes vs. VLMs as Runtime Monitors

Next, we study how our lightweight progress probe compares to general-purpose VLMs as runtime monitors.

Baselines. We compare Progress Probe (P^2) against the 4B-parameter Robometer [6]. We treat Robometer as a strong reference for task-progress estimation because it is trained on diverse robot data across hardware and simulation, including successful, failed, and intermediate-progress examples. Robometer provides two prediction heads: *Progress*, which

estimates the current task progress, and *Success*, which estimates the probability of task success and is primarily intended as a terminal signal. We evaluate both heads within the same runtime monitoring framework defined in Sec. V.

To calibrate the Progress head, we run Robometer on the same 300 training trajectories used for Progress Probe and aggregate its predicted progress curves for each task, defining nominal progress from Robometer’s own predictions on successful trajectories. We evaluate two variants: Robometer-ZS (zero-shot) and Robometer-FT, fine-tuned on the same success-and-failure dataset used to train SAFE.

Results: Progress Probes are Fast & Competitive with VLMs. Fig. 2 and Table I show that P^2 and Robometer-FT achieve comparable success-failure discriminability. However, P^2 fine-tunes only 2K parameters compared with 4B parameters for Robometer-FT and runs over $10,000\times$ faster at inference time. Thus, P^2 achieves performance comparable to a large VLM while being substantially more compact and computationally efficient.

C. Does Robot Action Data Matter for Progress Probing?

Unlike VLMs, which are trained only on images and text, VLAs are additionally fine-tuned on robot *action* data. We ask whether this action-grounded training yields internal activations that better encode task progress by comparing the performance of progress probes trained on VLA activations and those trained on VLM activations.

Methods. We compare our progress probe trained on VLA activations ($P^2_{\pi_{0.5}}$) to PaliGemma-Probe (P^2_{Pali}), Robometer-ZS-Probe ($P^2_{\text{Robo-ZS}}$), and Robometer-FT-Probe ($P^2_{\text{Robo-FT}}$). The latter are trained on the corresponding VLM’s activations in the same process we trained the original progress probe. We use the 3B PaliGemma which is the backbone of $\pi_{0.5}$.

Metrics. We measure the progress prediction from the probe using R^2 on the success rollouts from the evaluation data split. We also measure the success/failure AUROC of the probes as in the previous experiment.

Results: Robot Action Data Improves Progress Probe Performance. As seen in Fig. 3, $P^2_{\pi_{0.5}}$ performs the best at predicting progress on successful rollouts with an R^2 of 0.83 compared to $P^2_{\text{Robo-ZS}}$ ($R^2 = 0.63$), $P^2_{\text{Robo-FT}}$ ($R^2 = 0.76$), and P^2_{Pali} ($R^2 = 0.69$). We find that models that see *visuals* of robots in training (like in Robometer) do not lead to improvement over a standard VLM when it comes to predicting progress. Additionally, when it comes to monitoring, $P^2_{\pi_{0.5}}$ achieves an AUROC of 0.93 on the simulation evaluation set, outperforming of $P^2_{\text{Robo-ZS}}$ (AUROC: 0.61), $P^2_{\text{Robo-FT}}$ (AUROC: 0.81), and P^2_{Pali} (AUROC: 0.88). We hypothesize this difference is caused by the VLA training objective, i.e., learning to predict actions has caused the policy to develop internal representations that can better detect when the policy is entering failure. We argue this result points to the importance of not just robot data, but action data, in understanding whether the policy will succeed or fail in its current rollout.

Method	Block Stack	Knife Put	Stacked Utensil	Candy Bag	Coffee Bag	Coffee Bean	Average AUROC
FAIL-Detect (lopO)	0.40	0.46	0.59	0.67	0.49	0.42	0.54
FIPER (RND-OE)	0.61	0.82	0.74	0.50	0.60	0.52	0.61
FIPER (ACE)	0.61	0.68	0.62	0.50	0.60	0.50	0.57
STAC	0.59	0.57	0.37	0.31	0.57	0.34	0.45
SAFE-MLP	0.60	0.80	0.60	0.72	0.39	0.38	0.73
SAFE-LSTM	0.70	0.70	0.57	0.76	0.50	0.33	0.58
FAIL-Detect (logpZO)	0.47	0.94	0.79	0.90	0.72	0.74	0.69
Robometer (progress)	0.53	0.51	0.73	0.94	0.61	0.66	0.70
Robometer (success)	0.33	0.33	0.30	0.46	0.24	0.25	0.41
Robometer FT (progress)	0.63	0.56	0.88	0.95	0.66	0.53	0.75
Robometer FT (success)	0.32	0.43	0.28	0.39	0.20	0.18	0.34
Progress Probe (P^2)	0.87	0.80	0.81	0.92	0.66	0.26	0.78

TABLE III: **Hardware: All-task.** State-level failure-detection AUROC is reported for each task. The rightmost column reports the average performance across all six tasks.

Method	Block Stack	Knife Put	Stacked Utensil	Candy Bag	Coffee Bag	Coffee Bean	Average AUROC
FAIL-Detect (lopO)	0.51	0.66	0.54	0.55	0.60	0.49	0.56
FIPER (RND-OE)	0.40	0.83	0.76	0.62	0.51	0.56	0.61
FIPER (ACE)	0.61	0.68	0.62	0.50	0.60	0.50	0.59
STAC	0.59	0.57	0.37	0.31	0.57	0.34	0.46
SAFE-MLP	0.60	0.78	0.64	0.75	0.53	0.31	0.60
SAFE-LSTM	0.54	0.64	0.56	0.70	0.51	0.14	0.51
FAIL-Detect (logpZO)	0.48	0.80	0.77	0.83	0.50	0.75	0.69
Progress Probe (P^2)	0.87	0.75	0.84	0.94	0.60	0.84	0.81

TABLE IV: **Hardware: Unseen Task.** State-level failure detection AUROC is reported for each task. The rightmost column reports the average performance across all six tasks.

VII. EXPERIMENTS: HARDWARE

Finally, we study progress probes on hardware with a suite of manipulation tasks of increasing physical complexity.

Setup. Our hardware setup is a Franka Research 3 robotic arm with the DROID [25] camera setup. Our VLA is $\pi_{0.5}$ [1]. We use the $\pi_{0.5}$ -DROID checkpoint and finetune it on our 6 hardware tasks with 60 trajectories per task (360 trajectories total), from StressDream [26]. Our hardware tasks are:

- 1) **Candy Bag:** Pick and place an open candy bag into a tan bin without spilling the candy.
- 2) **Coffee Bag:** Pick and place an open coffee bag into a tan bin without spilling the coffee beans.
- 3) **Coffee Bean:** Pour coffee beans into the bowl without spilling the coffee beans.
- 4) **Knife Put:** Pick and place the knife into a bowl.
- 5) **Block Stack:** Pick and place the orange Jenga block onto the blue Jenga block.
- 6) **Stacked Utensil:** Pick up the blue spoon without pushing the brown spoon off the plate.

Training. Our probe training data is obtained by passing 360 hardware trajectories (from the VLA fine-tuning dataset) through the policy to get corresponding activation data. We label the data with the progress labels as in Eq. 2. The probe is trained on 80/20 splits for training and validation.

Evaluation. We evaluate all methods using 120 rollouts from the VLA policy (20 rollouts per task) with a 50/50

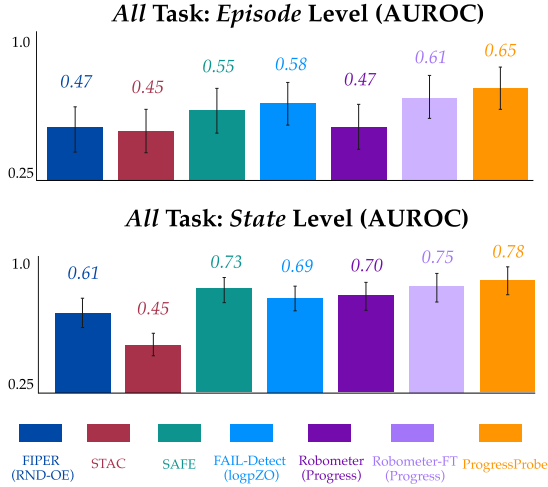


Fig. 4: **Hardware: Runtime Monitoring.** Episode-level and state-level AUROC for failure detection. P^2 achieves strong discriminability at both levels for runtime monitoring.

split for success and failure. For all failure rollouts, we manually label where the failure occurs in the episode and mark the remainder of the trajectory as a failure for frame-level evaluation.

Baselines. We compare Progress Probe (P^2) against the same runtime monitoring baselines used in simulation, adopting the same implementations and monitor categories: **input-level** methods (FIPER (RND-OE) [7, 24] and FAIL-Detect (logPO) [17]), **feature-level** methods (FAIL-Detect (logpZO) [17], SAFE-MLP, and SAFE-LSTM [4]), and **output-level** methods (FIPER (ACE) and STAC [5]). We additionally compare against Robometer [6] in both zero-shot and fine-tuned variants. As in simulation, SAFE and Robometer-FT use additional failure data, while the remaining monitors, including P^2 , are trained using successful trajectories only.

A. Discriminability by Failure Detection

We test if P^2 scales to monitoring a real-world VLA policy, evaluating how well P^2 distinguishes successful from failed execution when deployed on the real robot.

Method. We evaluate each baseline on the hardware evaluation dataset and ask the monitor to determine (a) if the entire episode is a successful or failed rollout via max aggregation and (b) given a trajectory, if the current time-step is a successful state or failure state.

Result: P^2 Better Discriminates Success and Failure at Both Episode and State Levels. Fig. 4 shows that P^2 achieves higher Episode- and State-level AUROC than all baselines, including Robometer-FT and SAFE, which require additional failure trajectories and labels for training. Fig. 5 further shows that P^2 qualitatively separates successful and failed executions: predicted progress falls behind the nominal trajectory and crosses the detection boundary near the onset of failure, while remaining within the nominal range on successful rollouts.

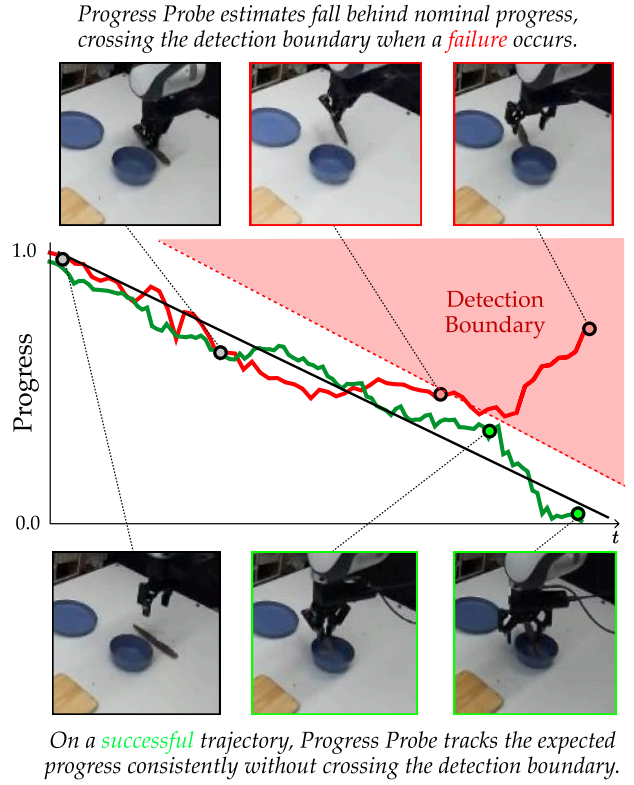


Fig. 5: **Hardware: Qualitative.** Red curves show failed trajectories and P^2 predictions, while green curves show predictions on successful trajectories for the Knife Put task. P^2 correctly identifies failures while remaining within the nominal range on successful executions, demonstrating reliable separation between successful and failed behavior.

Table III further shows that P^2 performs strongly across hardware tasks, achieving the highest AUROC on Block Stack (0.87) and Stacked Utensil (0.81), while remaining competitive on the remaining tasks. P^2 underperforms on more challenging tasks such as Coffee Bean, where failures can arise stochastically despite in-distribution actions and may be visually subtle due to the small image region associated with the failure. We hypothesize that, in such cases, the VLA activations may not change sufficiently to provide a strong failure signal. Mirroring the simulation results, **feature-level** monitors generally perform strongly on hardware, while fine-tuned Robometer achieves comparable performance on several tasks. The task-dependent variation across methods suggests that internal VLA features and visual task-progress cues provide complementary signals for runtime failure monitoring.

B. How Well Do P^2 Monitors Generalize to Unseen Tasks?

Next, we test the generalizability of progress probes and baseline runtime monitors to unseen tasks in hardware.

Method. Given that we have 6 tasks, we choose 3 tasks randomly to be unseen and repeat till all $\binom{6}{3} = 20$ combinations are exhausted. We re-train our method and all baselines on the 3 seen tasks, and evaluate on the unseen tasks' rollouts

from the evaluation set at the State level. We do not include Robometer since this VLM is typically used zero-shot, so all tasks would be “unseen” at all times.

Result: P^2 Generalizes Better to Unseen Tasks. Table IV shows that P^2 generalizes strongly to held-out tasks, achieving the highest State AUROC of 0.81 averaged across all $\binom{6}{3} = 20$ seen/unseen task splits. We hypothesize that this generalization arises from probing internal VLA activations, which encode shared representations learned across diverse tasks and hardware interactions. Per-task results further show that P^2 consistently generalizes well across the held-out tasks, suggesting that Progress Probe captures task-progress structure shared across diverse manipulation settings.

VIII. CONCLUSION AND LIMITATIONS

In this work, we introduced *Progress Probes* (P^2), a lightweight approach to runtime monitoring that linearly decodes task progress from a VLA’s internal activations. Across simulation and hardware experiments, P^2 achieves competitive failure-monitoring performance while being substantially more computationally efficient than existing VLM-based monitors. These results suggest that task progress encoded within VLA activations provides a simple yet effective signal for runtime failure monitoring.

Limitations. Despite promising results across simulation and hardware, several limitations remain. First, our experiments focus on $\pi_{0.5}$ [1], and future work should investigate how Progress Probes transfer across different VLA families, such as GR00T [2] and MolmoAct [27], whose architectures and training recipes induce different internal representations. Second, we primarily probe activations from the VLM backbone of the VLA. Many recent VLAs additionally employ continuous action experts, such as diffusion or flow-matching models; investigating whether their internal activations provide complementary signals for runtime monitoring is an important direction for future work.

REFERENCES

- [1] Physical Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia *et al.*, “ $\pi_{0.5}$: A vision-language-action model with open-world generalization,” in *Conference on Robot Learning (CoRL)*, 2025.
- [2] NVIDIA, J. Bjorck, F. Castañeda, N. Cherniadev, X. Da *et al.*, “GR00T N1: An open foundation model for generalist humanoid robots,” *arXiv preprint arXiv:2503.14734*, 2025.
- [3] J. Kwok, X. Zhang, M. Xu, Y. Liu, A. Mirhoseini *et al.*, “Scaling verification can be more effective than scaling policy learning for vision-language-action alignment,” in *European Conference on Computer Vision (ECCV)*, 2026.
- [4] Q. Gu, Y. Ju, S. Sun, I. Gilitshenski, H. Nishimura *et al.*, “SAFE: Multitask failure detection for vision-language-action models,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.
- [5] C. Agia, R. Sinha, J. Yang, Z.-a. Cao, R. Antonova *et al.*, “Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress,” in *Conference on Robot Learning (CoRL)*, 2024.
- [6] A. Liang, Y. Korkmaz, J. Zhang, M. Hwang, A. Anwar *et al.*, “Robometer: Scaling general-purpose robotic reward models via trajectory comparisons,” in *Robotics: Science and Systems (RSS)*, 2026.
- [7] R. Römer, A. Kobras, L. Worbis, and A. P. Schoellig, “Failure prediction at runtime for generative robot policies,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.
- [8] H. Buurmeijer, C. A. Alonso, A. Swann, and M. Pavone, “Observing and controlling features in vision-language-action models,” *arXiv preprint arXiv:2603.05487*, 2026.
- [9] H. Lu, H. Li, P. S. Shahani, S. Herbers, and M. Scheutz, “Probing a vision-language-action model for symbolic states and integration into a cognitive architecture,” in *IEEE International Conference on Artificial Intelligence and Data Analytics (ICAD)*, 2025.
- [10] T. Mikolov, W.-t. Yih, and G. Zweig, “Linguistic regularities in continuous space word representations,” in *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2013.
- [11] K. Park, Y. J. Choe, and V. Veitch, “The linear representation hypothesis and the geometry of large language models,” in *International Conference on Machine Learning (ICML)*, 2024.
- [12] A. Zou, L. Phan, S. L. Chen, J. Campbell, P. Guo *et al.*, “Representation engineering: A top-down approach to AI transparency,” *arXiv preprint arXiv:2310.01405*, 2023.
- [13] E. Hubinger, C. Denison, J. Mu, M. Lambert, M. Tong *et al.*, “Sleeper agents: Training deceptive LLMs that persist through safety training,” *arXiv preprint arXiv:2401.05566*, 2024.
- [14] M. MacDiarmid, T. Maxwell, N. Schiefer, J. Mu, J. Kaplan *et al.*, “Simple probes can catch sleeper agents,” *Anthropic Alignment Science Blog*, 2024.
- [15] B. Häon, K. Stocking, I. Chuang, and C. Tomlin, “Mechanistic interpretability for steering vision-language-action models,” in *Conference on Robot Learning (CoRL)*, 2025.
- [16] A. Swann, L. McGranahan, H. Buurmeijer, M. Kennedy, and M. Schwager, “Sparse autoencoders reveal interpretable and steerable features in VLA models,” in *Conference on Robot Learning (CoRL)*, 2026.
- [17] C. Xu, T. K. Nguyen, E. Dixon, C. Rodriguez, P. Miller *et al.*, “Can we detect failures without failure data? Uncertainty-aware runtime failure detection for imitation learning policies,” in *Robotics: Science and Systems (RSS)*, 2025.
- [18] J. Kwok, C. Agia, R. Sinha, M. Foutter, S. Li *et al.*, “RoboMonkey: Scaling test-time sampling and verification for vision-language-action models,” in *Conference on Robot Learning (CoRL)*, 2025.
- [19] Y. Wu, R. Tian, G. Swamy, and A. Bajcsy, “From foresight to forethought: VLM-in-the-loop policy steering via latent alignment,” in *Robotics: Science and Systems (RSS)*, 2025.
- [20] Y. Belinkov, “Probing classifiers: Promises, shortcomings, and advances,” *Computational Linguistics*, vol. 48, no. 1, 2022.
- [21] L. Beyer, A. Steiner, A. S. Pinto, A. Kolesnikov, X. Wang *et al.*, “PaliGemma: A versatile 3B VLM for transfer,” *arXiv preprint arXiv:2407.07726*, 2024.
- [22] G. Alain and Y. Bengio, “Understanding intermediate layers using linear classifier probes,” in *International Conference on Learning Representations (ICLR) Workshop Track*, 2017.
- [23] S. Zhang, Z. Xu, P. Liu, X. Yu, Y. Li *et al.*, “VLABench: A large-scale benchmark for language-conditioned robotics manipulation with long-horizon reasoning tasks,” in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.
- [24] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, “Exploration by random network distillation,” in *International Conference on Learning Representations (ICLR)*, 2019.
- [25] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari *et al.*, “DROID: A large-scale in-the-wild robot manipulation dataset,” in *Robotics: Science and Systems (RSS)*, 2024.
- [26] J. Seo, S. Veer, R. Tian, W. Ding, A. Sharma, K. Leung, E. Schmerling, M. Pavone, and A. Bajcsy, “Stressdream: Steering video world models for robust policy evaluation and improvement,” in *Conference on Robot Learning (CoRL)*, 2026.
- [27] H. Fang, J. Duan, D. Clay, S. Wang, S. Liu, W. Huang, X. Fan, W.-C. Tsai, S. Chen, Y. R. Wang *et al.*, “Molmoact2: Action reasoning models for real-world deployment,” *arXiv preprint arXiv:2605.02881*, 2026.