

---

# SAFE: Multitask Failure Detection for Vision-Language-Action Models

---

**Qiao Gu<sup>1,2,3</sup> Yuanliang Ju<sup>1,2,3</sup> Shengxiang Sun<sup>1,2</sup> Igor Gilitschenski<sup>1,2,3</sup>**

**Haruki Nishimura<sup>4</sup> Masha Itkina<sup>4</sup> Florian Shkurti<sup>1,2,3</sup>**

<sup>1</sup>University of Toronto (UofT), <sup>2</sup>UoFT Robotics Institute,

<sup>3</sup>Vector Institute, <sup>4</sup>Toyota Research Institute (TRI)

q.gu@mail.utoronto.ca

## Abstract

While vision-language-action models (VLAs) have shown promising robotic behaviors across a diverse set of manipulation tasks, they achieve limited success rates when deployed on novel tasks out-of-the-box. To allow these policies to safely interact with their environments, we need a failure detector that gives a timely alert such that the robot can stop, backtrack, or ask for help. However, existing failure detectors are trained and tested only on one or a few specific tasks, while VLAs require the detector to generalize and detect failures also in unseen tasks and novel environments. In this paper, we introduce the multitask failure detection problem and propose *SAFE*, a failure detector for generalist robot policies such as VLAs. We analyze the VLA feature space and find that VLAs have sufficient high-level knowledge about task success and failure, which is generic across different tasks. Based on this insight, we design *SAFE* to learn from VLA internal features and predict a single scalar indicating the likelihood of task failure. *SAFE* is trained on both successful and failed rollouts, and is evaluated on unseen tasks. *SAFE* is compatible with different policy architectures. We test it on OpenVLA,  $\pi_0$ , and  $\pi_0$ -FAST in both simulated and real-world environments extensively. We compare *SAFE* with diverse baselines and show that *SAFE* achieves state-of-the-art failure detection performance and the best trade-off between accuracy and detection time using conformal prediction. More qualitative results can be found at <https://vla-safe.github.io/>.

## 1 Introduction

Recently, scaling up robot manipulation datasets has enabled the development of large vision-language-action (VLA) models, which are generalist manipulation policies that can follow language instructions and accomplish a wide range of tasks [1–6]. However, when VLAs are directly deployed on unseen tasks without collecting additional demonstrations and finetuning the model, they still suffer from limited success rates and a wide range of failure modes. This has been demonstrated by evaluations in recent work [2, 7, 4]: while their success rates on seen tasks are as high as 80-90%, those on unseen ones drop to 30-60% out-of-the-box. Therefore, to safely and reliably deploy VLA policies in the real world, it is important to promptly detect their potential failures.

Most existing failure detection methods train a separate failure detector for each task, and evaluate the detector only on that task [8–16]. While these methods work well for specialist policies, they do not suit generalists like VLAs. VLAs are designed to accomplish diverse tasks and may frequently encounter novel task instructions and unseen environments during deployment. In such cases, it is impractical to exhaustively collect rollouts and train a failure detector for every new task. Some recent works introduce task-generic failure detectors, but they either require sampling multiple actions [17]

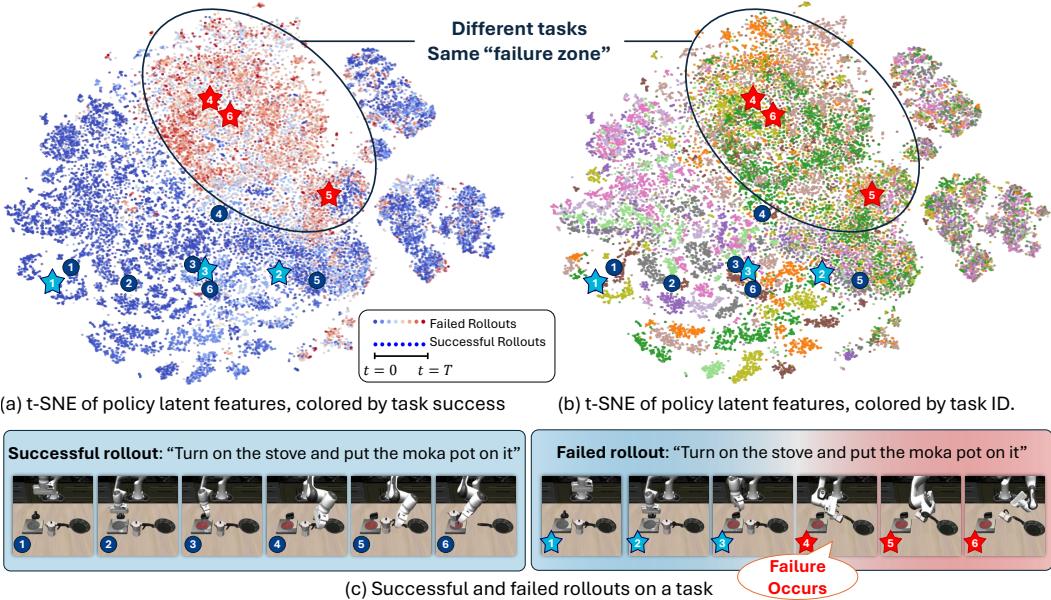


Figure 1: The VLA’s internal features capture high-level information about task success and failure. When the VLA is failing, even though from different tasks, the features fall in the same “failure zone”. This motivates *SAFE*, an efficient multitask failure detector that is based on VLA internal features and can generalize to unseen tasks. Plot (a) visualizes latent features of  $\pi_0$ -FAST on LIBERO-10 [20] using t-SNE [21]. For successful rollouts, their features are colored in blue; for failed rollouts, features are colored in the blue-red gradient based on timesteps. Plot (b) visualizes the same set of t-SNE features, colored by task ID. In (c), we show two example rollouts over time and mark their corresponding projected features in (a) and (b).

or need to query a large VLM [18, 19], which poses significant inference overhead for large VLAs in the real world. This motivates the need for an *efficient* and *multitask* failure detector that can generalize to unseen tasks zero-shot and detect failures in a timely manner during the on-policy rollout of the VLA.

In this paper, we focus on the *multitask* failure detection problem. This setting evaluates failure detection performance of a VLA policy without collecting rollouts or finetuning the failure detector on unseen tasks. To our knowledge, such multitask failure detection ability for VLAs has not been shown in the literature. To tackle this problem, we study the internal features of VLAs and find that they capture high-level knowledge about task success and failure. As shown in Fig. 1, such knowledge can effectively distinguish failed rollouts from successful ones and is generic across different tasks.

Based on this insight, we introduce *SAFE*, a ScAlable Failure Estimation method that scales across diverse tasks for generalist policies like VLAs. *SAFE* takes in a VLA’s internal features and regresses them to a single scalar indicating the likelihood of failure. By training on successful and failed rollouts of multiple tasks, *SAFE* learns to identify task-generic representations for failure detection. To determine the threshold for failure detection, we adopt the functional conformal prediction (CP) [22, 8] framework and calibrate the prediction band on the seen tasks. We conduct failure detection experiments on OpenVLA [2],  $\pi_0$  [4] and  $\pi_0$ -FAST [5], in both simulation and the real world. For evaluation, we adapt diverse baseline failure detection methods from both the LLM literature [23, 24] and the robot learning literature [8, 17] onto VLAs. *SAFE* and baselines are evaluated on both training tasks and a set of held-out tasks. Experiments show that *SAFE* outperforms other existing baselines and achieves the best trade-off between accuracy and timeliness for failure detection. The contributions of our paper can be summarized as follows:

- We analyze the VLA feature space and show that across different task instructions and environments, VLA’s internal features are separated for successful and failed rollouts.
- We propose *SAFE*, a multitask failure detector designed for generalist robot policies. By operating on latent features, training on multiple tasks, and using conformal prediction methods, *SAFE* shows generalization ability in detecting failures on unseen tasks.

- We evaluate *SAFE* and diverse baselines on several recent large VLA models in both simulation and the real world. Experiments show that *SAFE* outperforms baselines and achieves state-of-the-art (SOTA) performance.

## 2 Related Work

### 2.1 Vision-Language-Action Models

Recent advances in large-scale machine learning and the availability of extensive robot demonstration datasets have paved the way for VLA models [1–4, 7, 25, 26, 6]. These generalist robotic policies are initialized from pretrained large-scale VLMs [27–29], and thus inherit the strong ability to understand diverse semantic concepts from both images and language. They are augmented with an action head that produces continuous control signals, through per-step binning [1, 7, 2, 25], diffusion networks [3, 4, 30, 31] or frequency-space tokenization [5]. These VLAs are then trained on vast robotic datasets covering a wide array of tasks [32–34]. As a result, VLAs can successfully perform familiar tasks in new environments and even tackle previously unseen tasks when provided with novel language instructions. Nevertheless, significant variability in real-world deployments and the challenging domain gaps between training and testing environments continue to hinder VLA performance. Most state-of-the-art VLA models achieve success rates between 30% and 60% when evaluated out-of-the-box on real robots with unseen task instructions [2, 32, 4]. These limitations highlight the need for robust multitask failure detection methods tailored to generalist VLA models.

### 2.2 Failure Detection in Robot Manipulation

Monitoring failures is critical when deploying robotic policies in real-world environments, as even minor errors can result in hazardous conditions [35–37]. The literature on failure detection in robot learning can be broadly divided into unsupervised out-of-distribution (OOD) detection [8–11] and supervised failure detection [9, 12–16]. OOD detection-based methods treat successful executions as the in-domain baseline and consider any deviation from this norm as a failure. However, the assumption that any unseen scenario constitutes a failure is overly restrictive for generalist VLAs, which may frequently encounter unseen tasks at test time. These unseen tasks are likely different from the in-domain training data but should not be simply treated as failures. Our proposed method, *SAFE*, falls within the supervised failure detection category, leveraging both successful and failed rollouts to train a failure classifier. Yet *SAFE* trains a single unified failure detector for all tasks, while other existing ones typically require training and calibration of separate classifiers for each task and have not yet demonstrated effectiveness on generalist policies like VLAs. Some recent works have explored the multitask failure detection task by designing action consistency scores [17] or instruction-finetuning a VLM [18, 19], but they require either sampling multiple actions or querying a large VLM, which poses significant overhead for controlling robots in real time.

Recently, FAIL-Detect [8] conducted a systematic evaluation of various failure detection methods, including OOD detection-based approaches [38, 39, 9], smoothness-based techniques [40], and consistency-based strategies [17]. Their experiments indicate that the best performance was achieved by LogpZO [38] and RND [39], both using a learned network to model data distribution and detect OOD states. However, their evaluation is limited to only single-task policies, and our evaluation in multitask setting shows that their best performing RND and LogpZO methods suffer from overfitting to the training tasks.

### 2.3 Uncertainty Quantification for LLM

Although LLMs and VLMs have demonstrated remarkable understanding and generative capabilities across various tasks, they are prone to producing hallucinated responses [41–43]. Numerous methods have been developed for uncertainty quantification (UQ) in LLMs/VLMs. Token-level uncertainty quantification methods estimate uncertainty by analyzing the probability distribution over each generated token to assess the likelihood of an entire response [44–46]. In contrast, semantic-similarity methods generate multiple responses to the same query and evaluate their semantic alignment [24, 47, 48]; a higher variance among responses typically signals low confidence. Since vision-language-action models (VLAs) share the generative nature and transformer architecture of LLMs/VLMs, we adapt these UQ methods to VLAs as promising baselines and evaluate their

performance on failure detection. Recent research has also explored the internal latent space of LLMs for hallucination detection [49–55]. These methods train a classifier on internal latent features to distinguish between truthful and hallucinated outputs, paralleling supervised failure detection techniques in robotics. This approach has proven to be simple, efficient, and effective for UQ in LLMs. In our study, we investigate its application to large VLA policies and observe promising performance in robotic tasks.

### 3 Problem Formulation

This work aims to detect when a robot policy fails its task execution. Specifically, we aim for a multi-task failure detector that performs well when generalist VLAs perform novel tasks at inference time. At timestep  $t$ , a VLA is given an input observation  $\mathbf{o}_t$ , consisting of RGB images, natural language instruction, and current robot state, and outputs a control signal  $\mathbf{A}_t = [\mathbf{a}_t, \mathbf{a}_{t+1}, \dots, \mathbf{a}_{t+H-1}]$ , which is a chunk of actions for the next  $H$  timesteps. The first  $H'$  ( $H' \leq H$ ) actions in  $\mathbf{A}_t$  are executed, and then the VLA replans a new action sequence  $\mathbf{A}_{t+H'}$  at time  $t + H'$ . We denote the internal embedding vector within the VLA model at time  $t$  as  $\mathbf{e}_t$ . Some VLAs [1, 2, 4, 5] also decode a series of  $m$  tokens  $\mathbf{W}_t = [\mathbf{w}_t^1, \dots, \mathbf{w}_t^m]$  before converting them into the actual action vector. To train and evaluate failure detection models, we run the VLA on different tasks in simulation or the real world, collect the rollout trajectory  $\tau_i = \{(\mathbf{o}_t, \mathbf{e}_t, \mathbf{W}_t, \mathbf{A}_t)\}_{t=0, H', \dots, nH'}$  with time duration  $T = nH'$ , and annotate each rollout with a failure label  $y_i$  ( $y_i = 1$  if the robot fails to accomplish the task and  $y_i = 0$  if the robot succeeds). Note that for training, we only use the trajectory-level annotation  $y_i$ , and don't require knowing the exact timestep when the policy starts to fail. A failure detector receives the rollout information up to time  $t$  and predicts a failure score  $s_t$ , indicating the likelihood of task execution failure at time  $t$ . If  $s_t$  exceeds a threshold  $\delta_t$ , a failure flag is raised, and then either the task execution is aborted or a human monitor will step in and take over the control. In this work, we use conformal prediction [56] to calibrate the threshold  $\delta_t$ .

In experiments, we split all tasks into *seen* and *unseen* subsets, where rollouts from seen tasks are used for training  $\mathcal{D}_{\text{train}}$  and validation  $\mathcal{D}_{\text{eval-seen}}$ , and all rollouts from unseen tasks  $\mathcal{D}_{\text{eval-unseen}}$  are reserved for testing the cross-task generalization ability of failure detectors. Failure detectors are trained on  $\mathcal{D}_{\text{train}}$ , and evaluated on  $\mathcal{D}_{\text{eval-seen}}$  for hyperparameter tuning and in-domain performance, and tested on  $\mathcal{D}_{\text{eval-unseen}}$  for out-of-distribution generalization.

## 4 Method

### 4.1 Visual Analysis on VLA Latent Space

VLAs process multi-modal inputs and extract rich semantic information in their internal feature space. We hypothesize that these features also capture the high-level and abstract knowledge about task execution success/failure, by separating features from successful/failed rollouts into different regions. We study this hypothesis by visualizing the VLA features in Fig. 1, where we plot the internal features from  $\pi_0$ -FAST[5] when running the LIBERO-10 benchmark [20]. Fig. 1(a) demonstrates that when the VLA is failing, its internal features are grouped in the same region in the feature space (“failure zone”). Comparing Fig. 1(a) and Fig. 1(b), we can further see that although the features are extracted from different tasks with various instructions, objects and environments, when the VLA fails, its features fall in the same “failure zone”. Fig. 1(c) further illustrates how VLA’s features evolve in the feature space when VLA progresses temporally. From Fig. 1(c), we can see that failure rollout initially stays out of the “failure zone” when it progresses normally, and when the robot mistakenly drops the pot in the middle of execution and starts to fail, it steps into the “failure zone”. On the contrary, for the successful rollout, its features always stay out of the “failure zone”.

This visual analysis shows that the VLA’s internal features for succeeding and failing task executions are well separated in the feature space, and this separation is general across different tasks. Furthermore, during task execution, the features reflect how well the VLA performs on the current tasks in a timely manner. Inspired by this observation, we design *SAFE*, which uses the internal features of VLAs for failure detection.

Table 1: Failure detection results on simulation benchmarks, measured by area under ROC (ROC-AUC). “-” indicates that the failure detection method does not apply. Entries with gray background indicate the failure detection methods that sample 10 actions per inference timestep, while others use only 1 action. The **first** and **second** best-performing methods are colored in red and orange, respectively. Results are averaged over 3 random seeds with different splits of seen and unseen tasks.

	VLA Model Benchmark Eval Task Split	OpenVLA LIBERO		$\pi_0$ -FAST LIBERO		$\pi_0$ LIBERO		$\pi_0^*$ SimplerEnv		Average	
		Seen	Unseen	Seen	Unseen	Seen	Unseen	Seen	Unseen	Seen	Unseen
Token Unc.	Max prob.	50.25	53.83	61.32	69.44	-	-	-	-	55.79	61.64
	Avg prob.	44.05	51.58	52.46	58.04	-	-	-	-	48.26	54.81
	Max entropy	52.94	53.09	46.69	62.96	-	-	-	-	49.81	58.03
	Avg entropy	45.27	50.03	50.93	58.63	-	-	-	-	48.10	54.33
Embed. Distr.	Mahalanobis dist.	62.03	58.85	<b>93.56</b>	83.79	<b>77.12</b>	<b>74.31</b>	88.42	52.84	80.28	67.45
	Euclidean dist. k-NN	66.00	55.23	92.04	84.12	75.64	70.73	<b>89.73</b>	68.41	80.85	69.62
	Cosine dist. k-NN	67.09	69.45	92.09	<b>84.64</b>	75.76	70.31	<b>90.19</b>	71.32	81.28	73.93
	PCA-KMeans [9]	57.18	55.10	68.46	57.12	64.92	60.35	66.88	61.19	64.36	58.44
	RND [39]	52.57	46.88	88.67	81.57	71.92	69.44	85.07	65.89	74.56	65.95
	LogpZO [8]	61.57	52.91	91.52	83.07	76.80	73.23	88.79	74.66	79.67	70.97
Sample Consist.	Action total var.	62.76	65.43	76.95	74.50	77.20	75.18	68.41	67.94	71.33	70.76
	Trans. total var.	55.33	58.99	78.21	80.03	49.38	54.71	63.27	55.90	61.55	62.41
	Rot. total var.	47.85	55.30	80.87	77.29	52.94	61.06	58.07	62.10	59.93	63.94
	Gripper total var.	61.84	64.48	76.82	74.42	77.19	75.19	69.16	69.29	71.25	70.84
	Cluster entropy	50.16	51.44	80.22	80.53	76.19	72.12	68.25	73.66	68.71	69.44
Action Consist.	STAC [17]	-	-	83.07	85.31	46.55	47.91	60.74	62.21	63.45	65.14
	STAC-Single	-	-	85.46	81.16	68.46	69.39	68.71	70.40	74.21	73.65
SAFE (Ours)	SAFE-LSTM	<b>70.24</b>	<b>72.47</b>	<b>92.98</b>	<b>84.48</b>	<b>76.98</b>	71.09	88.85	<b>80.11</b>	<b>82.26</b>	<b>77.04</b>
	SAFE-MLP	<b>72.68</b>	<b>73.47</b>	90.06	80.44	73.50	<b>73.27</b>	89.50	<b>84.82</b>	<b>81.43</b>	<b>78.00</b>

## 4.2 Failure Detection by Feature Probing

We design *SAFE* to learn the abstract information from the VLA’s internal features and determine whether the task execution is failing. We extract the VLA’s hidden state vectors from the final layer, before being decoded to token logits [2, 5] or a velocity field [4]. We ablate different ways to aggregate the internal features into a single embedding vector  $e$ , and select the best one based on  $\mathcal{D}_{\text{eval-seen}}$  performance. Please refer to Appendix for details on VLA feature extraction.

The failure detector  $f(e_{0:t})$  takes as input the VLA’s features  $e_{0:t} = \{e_1, \dots, e_t\}$  up to the current timestep  $t$ , and is trained to predict  $s_t$ . We explore the two backbone designs for *SAFE*: a multi-layer perceptron ( $f_{\text{MLP}}$ ) and an LSTM [57] ( $f_{\text{LSTM}}$ ). Both models are designed to be simple (only one or two layers), to avoid overfitting and improve generalization ability on unseen tasks. For  $f_{\text{MLP}}$ , we use an MLP  $g(\cdot)$  to project  $e_t$  into a single scalar for each timestep  $t$  independently and accumulate the outputs as the failure score, i.e.  $f_{\text{MLP}}(e_{0:t}) = \sum_{\tau=1, \dots, t} \sigma(g(e_\tau))$ , where  $\sigma(\cdot)$  is a sigmoid function and therefore  $0 < s_t < t$ . To train the MLP model, we apply an L1 loss on all timesteps to push up the scores for failed rollouts and push down those for successful ones. Specifically,  $L_{\text{MLP}} = \sum_i [y_i \sum_t (t - s_t) + (1 - y_i) \sum_t s_t]$ , where index  $i$  iterates over all data points in  $\mathcal{D}_{\text{train}}$ .

For  $f_{\text{LSTM}}$ , we use an LSTM model to sequentially process the input stream of VLA’s features  $e_{0:t}$  and project the hidden state vector of LSTM into a scalar score. Specifically,  $f_{\text{LSTM}}(e_{0:t}) = \sigma(\text{LSTM}(e_{0:t}))$ , where a sigmoid function  $\sigma(\cdot)$  is applied to normalize the output score s.t.  $0 \leq s_t \leq 1$ . To train the LSTM model, we apply a binary cross entropy loss on all timesteps, i.e.  $L_{\text{LSTM}} = \sum_i \sum_t [y_i \log(s_t) + (1 - y_i) \log(1 - s_t)]$ .

## 4.3 Threshold Selection by Conformal Prediction

When the predicted failure score  $s_t$  exceeds the time-varying threshold  $\delta_t$ , we raise a failure flag. To determine  $\delta_t$  in a principled way, we adopt the functional conformal prediction (CP) framework [22]. Functional CP constructs a time-varying prediction band by leveraging the distribution of  $s_t$  observed in successful rollouts within a calibration set. Under the exchangeability assumption [58] and given a user-specified significance level  $\alpha$ , the CP band guarantees that, for a new successful rollout, its  $s_t$  will lie within this band at all times  $t$  with probability  $1 - \alpha$ . Conversely, if the score of a test rollout exits the band at time  $t$ , we can declare a failure with nominal confidence  $1 - \alpha$ .

Formally, given a time series of any scalar score  $s_t$  and a user-specified significance level  $\alpha \in (0, 1)$ , functional CP gives a distribution-free prediction band  $C_\alpha$ . Following [8], we adopt the one-sided time-varying CP band formulation, where  $C_\alpha$  is a set of intervals  $\{\text{lower}_t, \text{upper}_t : t = 1, \dots, T\}$ , where  $\text{lower}_t = -\infty$  and  $\text{upper}_t = \mu_t + h_t$ , with a time-varying mean  $\mu_t$  and a bandwidth  $h_t$ . This

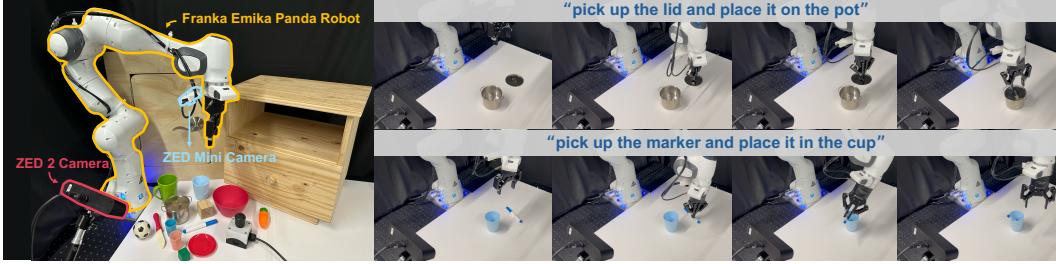


Figure 2: Illustration of real-world experiment setup (left) and example rollouts collected (right).

band is calibrated on successful rollouts in  $\mathcal{D}_{\text{eval-seen}}$ . Under mild assumptions [59, 60], for any new successful rollout,  $s_t < \mu_t + h_t$  holds for all  $t = 1, \dots, T$  with probability  $1 - \alpha$ . Intuitively, this gives a guarantee that the false positive rate of the failure detector (a failure flag is raised at any time during a successful rollout) is at most  $\alpha$ . We use  $\text{upper}_t$  as the failure flag threshold  $\delta_t$ , and more details about functional CP can be found in Appendix.

## 5 Experiments

### 5.1 Evaluation Benchmarks

**LIBERO** [20]: The LIBERO benchmark has been widely adopted for evaluating VLA models in simulation [2, 4–6]. Among the LIBERO task suites, the LIBERO-10 suite consists of 10 long-horizon tasks with diverse objects, layouts, and instructions, and is considered the most challenging one. Therefore, we use LIBERO-10 in our experiments and test OpenVLA [2],  $\pi_0$  [4] and  $\pi_0$ -FAST [5] on it. We adopt the model checkpoints that are finetuned on the LIBERO benchmark and released by their authors. In experiments, 3 out of 10 tasks are randomly picked and reserved as unseen tasks.

**SimplerEnv** [61]: SimplerEnv provides a high-fidelity simulation environment for manipulation policies, which are replicas of the demonstration data from RT-series [1, 7, 32] and BridgeData V2 [34]. On SimplerEnv, we test pretrained  $\pi_0$  models from a reproduction [62], which we denote as  $\pi_0^*$  in this paper. We train and evaluate the failure detection methods on the Google Robot embodiment [1] and on the WidowX embodiment [34], respectively. We exclude the “pick up coke” task because  $\pi_0^*$  rarely fails on it (success rate at 98%). This leaves 4 tasks for each embodiment, among which 3 tasks are seen and 1 task is unseen.

**Real-robot Experiments:** We deploy the  $\pi_0$ -FAST-DROID checkpoint<sup>1</sup> provided by [4, 5] on a Franka Emika Panda Robot. This checkpoint has been finetuned on the DROID dataset [33], and we do not further collect demonstrations or finetune the VLA model. We design 13 tasks and collect 30 successful and 30 failed rollouts for each task. The real-robot setup and example rollouts are visualized in Fig. 2. In experiments, 3 tasks out of 13 are randomly selected as unseen tasks.

### 5.2 Uncertainty Quantification Baselines

Estimating uncertainty in generated responses has been widely used to detect truthfulness or hallucination in LLMs [23, 24, 63, 43]. For VLAs, uncertainty in the generated actions may indicate a lack of ability to solve the given task, and thus correlates with task failures. Therefore, we first adapt the UQ methods from the LLM literature to VLAs and use them as failure detection baselines.

**Token uncertainty**-based methods aggregate the predictive uncertainty from each generated token. These methods are efficient, as they only require a single forward inference. Given the generated tokens  $\mathbf{W}_t = [\mathbf{w}_t^1, \dots, \mathbf{w}_t^m]$ , we denote the probability of sampling the token  $\mathbf{w}_t^i$  as  $p_i$  and the entropy over the distribution of the  $i^{\text{th}}$  token as  $H_i$ . We adopt the token-based uncertainty estimation methods proposed in [23] as follows:

$$\begin{aligned} \text{Token max prob.} &: \max_i (-\log p_i); & \text{Token avg prob.} &: -\frac{1}{m} \sum_i \log p_i; \\ \text{Token max entropy} &: \max_i H_i; & \text{Token avg entropy} &: \frac{1}{m} \sum_i H_i. \end{aligned}$$

<sup>1</sup><https://github.com/Physical-Intelligence/openpi>

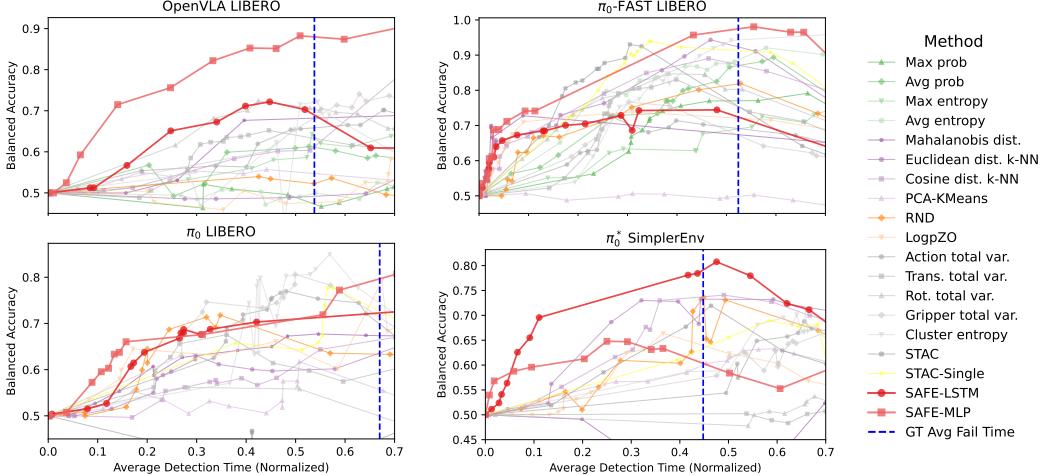


Figure 3: Performance of functional CP by varying significant level  $\alpha$  on  $\mathcal{D}_{\text{eval-unseen}}$ . The plots show the change of balanced accuracy (bal-acc) w.r.t. average detection time (T-det). Good failure detection methods should detect policy failures both accurately (high bal-acc) and proactively (lower T-det), and thus place curves towards the top left in each plot. We can see in all simulation experiments, the proposed *SAFE-LSTM* and *SAFE-MLP* perform better or on par with the best baselines. Note that baselines in gray require multiple action samples.

**Sample consistency**-based methods estimate uncertainty as the inconsistency within multiple generated sentences [23, 24, 63]. For VLA models, the output actions are continuous vectors, and we can measure inconsistency by their variance. Specifically, at time  $t$ , given  $K$  sampled actions  $\mathcal{A}_t = \{\mathbf{A}_t^k\}_{k=1,\dots,K}$ , we measure the uncertainty as the total variation over the set of vectors: *action total var.* =  $\text{trace}(\text{cov}(\mathcal{A}_t))$ . Similarly, we also compute variation for the translational (*trans. total var.*), rotational (*rot. total var.*), and gripper control (*gripper total var.*) components of  $\mathcal{A}_t$ .

Furthermore, inspired by semantic entropy [24], we define *cluster entropy* as  $\text{entropy}(\text{cluster}(\{\mathbf{A}_t^k\}_{k=1,\dots,K}))$ , where  $\text{cluster}(\cdot)$  generates an integer set, containing  $k$  cluster labels for the  $k$  actions and  $\text{entropy}(\cdot)$  measures the entropy of the integer set. UQ methods based on sample consistency are shown to perform better for LLM [24, 23], but it necessitates multiple inferences, which may not be practical for large VLAs that control robots in real time.

### 5.3 Failure Detection Baselines

**Embedding Distance:** We compare to baselines that directly use the distances in the feature space as failure scores. Specifically, instead of training a neural network, all VLA embeddings from  $\mathcal{D}_{\text{train}}$  are stored in the two feature sets,  $E_{\text{succ}}$  and  $E_{\text{fail}}$ , containing all VLA embeddings from successful and failed rollouts respectively. During evaluation on  $\mathcal{D}_{\text{eval-seen}}$  and  $\mathcal{D}_{\text{eval-unseen}}$ , failure scores are computed as  $s_t = d(\mathbf{e}_t, E_{\text{succ}}) - d(\mathbf{e}_t, E_{\text{fail}})$ , where  $d(\cdot, \cdot)$  measures the distance between a single vector and a set of vectors. Intuitively, if  $\mathbf{e}_t$  is far from  $E_{\text{succ}}$  and close to  $E_{\text{fail}}$ , it's more likely to fail. Following [17, 10], we ablate different types of distance, including Mahalanobis distance, and Euclidean and Cosine distance averaged over  $k$ -Nearest Neighbors of  $\mathbf{e}_t$ . We also compare to the PCA-KMeans distance measure from [9].

**Learned OOD Detector:** We adopt RND [39] and LogpZO [8], which are shown to be the best-performing OOD detection-based failure detectors by [8]. Both methods use a neural network  $f^{\text{OOD}}(\cdot)$  to model the embedding distribution from successful rollouts and return an OOD score for a new embedding. We adapt them to learn from both successful and failed rollouts by training two models,  $f_{\text{succ}}^{\text{OOD}}(\cdot)$  and  $f_{\text{fail}}^{\text{OOD}}(\cdot)$ , on  $E_{\text{succ}}$  and  $E_{\text{fail}}$  respectively. Similar to embedding distance baselines, the failure score is computed as  $s_t = f_{\text{succ}}^{\text{OOD}}(\mathbf{e}_t) - f_{\text{fail}}^{\text{OOD}}(\mathbf{e}_t)$ .

**Action Consistency:** STAC [17] detects policy failures by measuring the statistical distance on the overlapping segment of two consecutive predicted action chunks. As it requires sampling multiple actions from the policy ([17] uses 256 actions), it compromises real-time operation for real robots, because unlike relatively small diffusion policy networks, large VLAs are not optimized for parallel

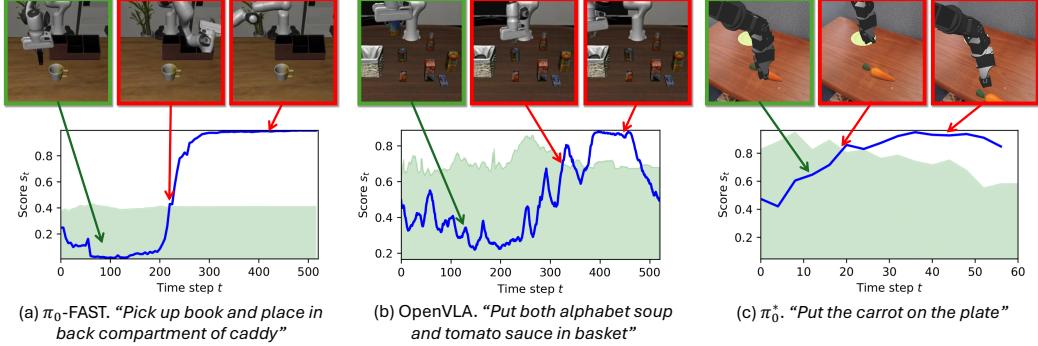


Figure 4: Failure scores predicted by *SAFE*-LSTM and the corresponding camera observations. The green areas show the functional CP band  $C_\alpha$ . Once failure scores exceed  $C_\alpha$ , a failure flag is raised. In (a),  $\pi_0$ -FAST policy misses the insertion, and its actions become unstable after that. In (b) and (c), OpenVLA and  $\pi_0^*$  miss the grasp but still proceed to the placing action, causing a failure detection. Note that these tasks are not seen when training *SAFE*-LSTM.

inference<sup>2</sup>. Therefore, we only test STAC in the simulation experiments with 10 sampled actions. We also adopt STAC-Single, a real-time version of STAC, which computes action inconsistency using only one sample from each inference timestep. Since OpenVLA only outputs one-step immediate action ( $H = 1$ ), STAC and STAC-single do not apply to it.

#### 5.4 Evaluation Protocol

We consider two types of evaluation. The first type evaluates how well  $s_t$  separates the successful and failed rollouts across all possible selections of  $\delta_t$ . Following the evaluation protocol widely adopted in the LLM UQ literature [49, 53, 23, 65], we use the area under ROC curve (ROC-AUC) metric. Furthermore, because a failure flag is raised whenever  $s_t$  exceeds  $\delta_t$ , a successful rollout (ground truth negative) becomes a false positive whenever  $s_t > \delta_t$ , and remains a true negative only if  $s_t \leq \delta_t$  for all time. Therefore, we consider the max-so-far score  $\bar{s}_T = \max_{\tau=1,\dots,T} s_\tau$  and compute the ROC-AUC metric based on  $\bar{s}_T$ , the maximum failure score throughout the entire rollout.

The second type of evaluation utilizes  $\delta_t = \text{upper}_t$  calibrated by functional CP in Section 4.3. By setting the significance level  $\alpha$ , we get a decisive positive/negative detection for each rollout. Following [17, 8], we consider the following metrics: true positive rate (TPR), false positive rate (FPR), balanced accuracy (bal-acc), and averaged detection time (T-det), where  $\text{Bal-Acc} = \frac{\text{TPR} + \text{TNR}}{2}$ . T-det is the relative timestep where  $s_t > \delta_t$  for the first time (if  $s_t$  never exceeds  $\delta_t$ , T-det becomes 1), averaged over all ground truth failed rollouts.

## 6 Results

### 6.1 How well do failure detectors distinguish failures from successes?

In Table 1 and Fig. 5 (a), we report the ROC-AUC metric based on  $\bar{s}_T$ , in simulation and real-world experiments, respectively. With a higher ROC-AUC metric, a failure detector achieves higher accuracy averaged over all possible thresholds. The tables show that **Token Unc.** methods have poor performance, which is aligned with findings in the LLM literature [24, 23]. On the other hand, the **Sample Consist.** and **STAC** [17] methods, which require multiple action samples, perform better and even achieve the best performance on unseen tasks in  $\pi_0$ -FAST LIBERO (**STAC**) and  $\pi_0$  LIBERO (**Gripper total var.**). However, as these methods require multiple action samples, they cause significant overhead for VLA models and thus are not applicable to real robots. **Embed. Distr.** methods perform well, achieving the best performance in two simulation benchmarks ( $\pi_0$  and  $\pi_0$ -FAST) and are the second best in the real world. This demonstrates that a VLA’s internal features are informative about task execution success/failure. The proposed **SAFE** methods perform better or

<sup>2</sup> $\pi_0$  is 1.52x slower and  $\pi_0$ -FAST is 2.21x slower to generate 10 action samples compared to 1 sample, tested on a single NVIDIA RTX 3090 GPU, with vmap optimization and JiT compilation in Jax [64]. For comparison, **SAFE** methods only cost negligible overhead (<1ms, or <1% of the inference time of  $\pi_0$  and  $\pi_0$ -FAST).

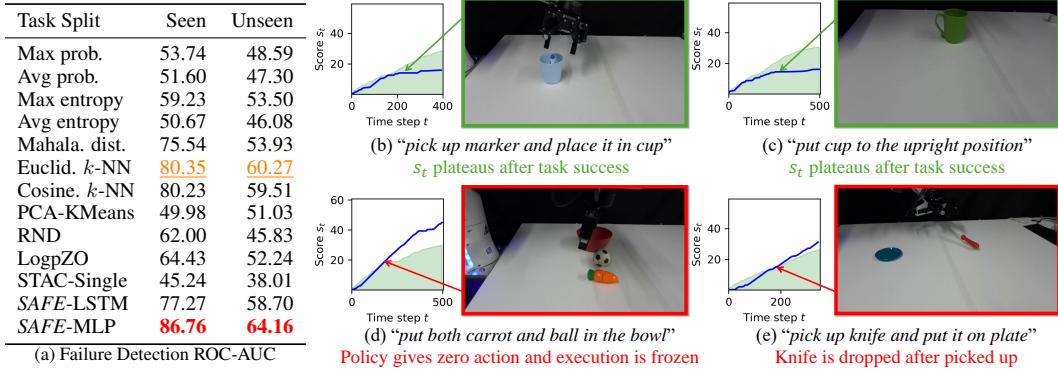


Figure 5: Quantitative results (a) and qualitative examples (b-e) from **SAFE-MLP** in the real world experiments. The ROC-AUC results are averaged over 5 random seeds with different task splits.

on par with the best baselines, consistently in all settings. Averaged across simulation benchmarks, **SAFE-MLP** and **SAFE-LSTM** have similar performance, both outperforming the best baseline by 4-5% on unseen tasks, while still achieving the best performance on seen tasks. On the real robot, **SAFE-MLP** achieves the best performance and **SAFE-LSTM** performs closely with the best baseline (Euclid.  $k$ -NN). Comparing **SAFE** with **Embed. Distr.** methods, we attribute the success of **SAFE** to its stronger ability to extract high-level abstract information from raw feature vectors through learned neural networks.

## 6.2 How do detection accuracy and detection time trade off using functional CP?

In Fig. 3, we use  $\mathcal{D}_{\text{eval-seen}}$  to calibrate functional CP band  $C_\alpha$  and evaluate on  $\mathcal{D}_{\text{eval-unseen}}$ . By varying the user-specified  $\alpha$ , we can adjust the conservativeness of the failure detectors and obtain a trade-off between accuracy (bal-acc) and detection time (T-det). To be a good failure detector, it should detect failures both accurately (higher bal-acc) and promptly (lower T-det), and thus have the curve rise toward the top-left corner in the plots of Fig. 3. As we can see from Fig. 3, the proposed **SAFE-MLP** and **SAFE-LSTM** perform the best on OpenVLA+LIBERO and  $\pi_0$ +SimplerEnv benchmarks, and on par with the best baseline on the other two benchmarks. We also manually annotate the ground truth (GT) failure timesteps (when a human thinks that failure happens or intervention is needed) for failed rollouts, and plot them as blue vertical lines in Fig. 3. Comparing **SAFE**'s performance with the GT fail time, we can see that **SAFE** can detect failures with high accuracy in the early stages of rollouts and potentially before the failure happens. This early detection allows early intervention for policy failures before they get stuck in execution or cause harm to the real-world environment.

## 6.3 What are detected failure modes and are they aligned with human intuition?

In Fig. 4 and Fig. 5(b-d), we visualize rollouts with the failure scores detected by **SAFE**. Fig. 4 demonstrates common failure modes in simulation, including imprecise insertion, unstable actions and missed grasps. Two successful rollouts on the real robot are shown in Fig. 5(b-c), where failure scores stop increasing after task completion. For the failed rollouts, the failure flag is raised after the policy is frozen (Fig. 5d) or the object slips out of the gripper (Fig. 5e). This aligns well with human intuition. Please refer to Appendix for video illustrations.

## 7 Conclusion

In this paper, we introduce the multitask failure detection problem for generalist VLA policies, where failure detectors are trained only on seen tasks and evaluated on unseen tasks. We analyze VLA's internal feature space and find that the internal features are separated for successful and failed rollouts. Based on this observation, we propose **SAFE**, a simple and efficient failure detection method by operating on the VLA's internal features. **SAFE** is evaluated on multiple VLAs in both simulation and the real world, and compared with diverse baselines. Experiments show that **SAFE** achieves SOTA results in failure detection, and aligns with human intuition.

**Limitations:** Most recent VLAs have shown capabilities in handling diverse modalities, controlling diverse embodiments and learning latent actions from non-robotic action-less video data [66, 67]. This paper only considers multitask failure detection for manipulation tasks, and it’s not clear how well the failure detectors generalize across embodiments, sim2real or to the action-less videos. Besides, *SAFE* only uses features from the last layer and how to effectively aggregate information across multiple layers of a VLA remains future work.

## Acknowledgments and Disclosure of Funding

The authors were partially supported by the Toyota Research Institute (TRI) and NSERC Discovery Grant. The authors thank Blerim Abdullai, Sebastian Aegidius, Jasper Gerigk, Ruthrash Hari, and Wei-Cheng Tseng for helpful discussions and feedback.

## References

- [1] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.
- [2] Moo Jin Kim, Karl Pertsch, Siddharth Karamchetti, 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 Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard, editors, *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pages 2679–2713. PMLR, 06–09 Nov 2025.
- [3] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yun-liang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.
- [4] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al.  $\pi_0$ : A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.
- [5] Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.
- [6] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.
- [7] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.
- [8] Chen Xu, Tony Khuong Nguyen, Emma Dixon, Christopher Rodriguez, Patrick Miller, Robert Lee, Paarth Shah, Rares Ambrus, Haruki Nishimura, and Masha Itkina. Can we detect failures without failure data? uncertainty-aware runtime failure detection for imitation learning policies. *arXiv preprint arXiv:2503.08558*, 2025.
- [9] 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 *8th Annual Conference on Robot Learning (CoRL)*, 2024.
- [10] Rohan Sinha, Amine Elhafsi, Christopher Agia, Matthew Foutter, Ed Schmerling, and Marco Pavone. Real-time anomaly detection and reactive planning with large language models. In *Robotics: Science and Systems*, 2024.

- [11] 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 Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1367–1378. PMLR, 08–11 Nov 2022.
- [12] 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)*, pages 4154–4161. IEEE, 2024.
- [13] Cem Gokmen, Daniel Ho, and Mohi Khansari. Asking for help: Failure prediction in behavioral cloning through value approximation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5821–5828. IEEE, 2023.
- [14] Annie Xie, Fahim Tajwar, Archit Sharma, and Chelsea Finn. When to ask for help: Proactive interventions in autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 35:16918–16930, 2022.
- [15] Rohan Sinha, Edward Schmerling, and Marco Pavone. Closing the loop on runtime monitors with fallback-safe mpc. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pages 6533–6540. IEEE, 2023.
- [16] Alec Farid, David Snyder, Allen Z. Ren, and Anirudha Majumdar. Failure prediction with statistical guarantees for vision-based robot control. In *Robotics: Science and Systems XVIII, New York City, NY, USA, June 27 - July 1, 2022*, 2022.
- [17] 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*, pages 689–723. PMLR, 2025.
- [18] 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. *arXiv preprint arXiv:2410.00371*, 2024.
- [19] Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. Vision-language models as success detectors. In *Conference on Lifelong Learning Agents*, pages 120–136. PMLR, 2023.
- [20] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.
- [21] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.
- [22] Jacopo Diquigiovanni, Matteo Fontana, Simone Vantini, et al. The importance of being a band: Finite-sample exact distribution-free prediction sets for functional data. *STATISTICA SINICA*, 1:1–41, 2024.
- [23] Yuheng Huang, Jiayang Song, Zhijie Wang, Shengming Zhao, Huaming Chen, Felix Juefei-Xu, and Lei Ma. Look before you leap: An exploratory study of uncertainty measurement for large language models. *arXiv preprint arXiv:2307.10236*, 2023.
- [24] Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023.
- [25] Haoyu Zhen, Xiaowen Qiu, Peihao Chen, Jincheng Yang, Xin Yan, Yilun Du, Yining Hong, and Chuang Gan. 3d-vla: A 3d vision-language-action generative world model. *arXiv preprint arXiv:2403.09631*, 2024.

- [26] Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Chilam Cheang, Ya Jing, Weinan Zhang, Huaping Liu, Hang Li, and Tao Kong. Vision-language foundation models as effective robot imitators. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*, 2024.
- [27] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- [28] Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, et al. Paligemma: A versatile 3b vlm for transfer. *arXiv preprint arXiv:2407.07726*, 2024.
- [29] Siddharth Karamcheti, Suraj Nair, Ashwin Balakrishna, Percy Liang, Thomas Kollar, and Dorsa Sadigh. Prismatic vlms: Investigating the design space of visually-conditioned language models. In *International Conference on Machine Learning*, pages 23123–23144. PMLR, 2024.
- [30] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In Kostas E. Bekris, Kris Hauser, Sylvia L. Herbert, and Jingjin Yu, editors, *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*, 2023.
- [31] Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 2025.
- [32] Open X-Embodiment Collaboration. Open x-embodiment: Robotic learning datasets and RT-X models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903, 2024. doi: 10.1109/ICRA57147.2024.10611477.
- [33] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, et al. DROID: A large-scale in-the-wild robot manipulation dataset. In *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*, 2024.
- [34] Homer Rich Walke, Kevin Black, Tony Z Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pages 1723–1736. PMLR, 2023.
- [35] Rohan Sinha, Apoorva Sharma, Somrita Banerjee, Thomas Lew, Rachel Luo, Spencer M Richards, Yixiao Sun, Edward Schmerling, and Marco Pavone. A system-level view on out-of-distribution data in robotics. *arXiv preprint arXiv:2212.14020*, 2022.
- [36] Rajesh Natarajan, Santosh Reddy, Subash Chandra Bose, HL Gururaj, Francesco Flammini, and Shanmugapriya Velmurugan. Fault detection and state estimation in robotic automatic control using machine learning. *Array*, 19:100298, 2023.
- [37] Quazi Marufur Rahman, Peter Corke, and Feras Dayoub. Run-time monitoring of machine learning for robotic perception: A survey of emerging trends. *IEEE Access*, 9:20067–20075, 2021.
- [38] Chen Xu, Xiuyuan Cheng, and Yao Xie. Normalizing flow neural networks by jko scheme. *Advances in Neural Information Processing Systems*, 36:47379–47405, 2023.
- [39] Nantian He, Shaohui Li, Zhi Li, Yu Liu, and You He. Rediffuser: Reliable decision-making using a diffuser with confidence estimation. In *International Conference on Machine Learning*, pages 17921–17933. PMLR, 2024.
- [40] Sivakumar Balasubramanian, Alejandro Melendez-Calderon, Agnes Roby-Brami, and Etienne Burdet. On the analysis of movement smoothness. *Journal of neuroengineering and rehabilitation*, 12:1–11, 2015.
- [41] Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*, 2024.

- [42] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12):248:1–248:38, 2023.
- [43] Ola Shorinwa, Zhiting Mei, Justin Lidard, Allen Z Ren, and Anirudha Majumdar. A survey on uncertainty quantification of large language models: Taxonomy, open research challenges, and future directions. *arXiv preprint arXiv:2412.05563*, 2024.
- [44] Chen Ling, Xujiang Zhao, Wei Cheng, Yanchi Liu, Yiyou Sun, Xuchao Zhang, Mika Oishi, Takao Osaki, Katsushi Matsuda, Jie Ji, Guangji Bai, Liang Zhao, and Haifeng Chen. Uncertainty decomposition and quantification for in-context learning of large language models. In *2024 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2024. URL <https://openreview.net/forum?id=0q1b1DnUOP>.
- [45] Yijun Xiao and William Yang Wang. On hallucination and predictive uncertainty in conditional language generation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2734–2744, 2021.
- [46] Andrey Malinin and Mark J. F. Gales. Uncertainty estimation in autoregressive structured prediction. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.
- [47] Zhen Lin, Shubhendu Trivedi, and Jimeng Sun. Generating with confidence: Uncertainty quantification for black-box large language models. *Trans. Mach. Learn. Res.*, 2024, 2024. URL <https://openreview.net/forum?id=DWkJCSxKU5>.
- [48] Juhai Chen and Jonas Mueller. Quantifying uncertainty in answers from any language model and enhancing their trustworthiness. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5186–5200, 2024.
- [49] Amos Azaria and Tom Mitchell. The internal state of an llm knows when it’s lying. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 967–976, 2023.
- [50] Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. *Advances in Neural Information Processing Systems*, 36:41451–41530, 2023.
- [51] Collin Burns, Haotian Ye, Dan Klein, and Jacob Steinhardt. Discovering latent knowledge in language models without supervision. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.
- [52] Xuefeng Du, Chaowei Xiao, and Sharon Li. Haloscope: Harnessing unlabeled llm generations for hallucination detection. *Advances in Neural Information Processing Systems*, 37:102948–102972, 2024.
- [53] Jannik Kossen, Jiatong Han, Muhammed Razzak, Lisa Schut, Shreshth Malik, and Yarin Gal. Semantic entropy probes: Robust and cheap hallucination detection in llms. *arXiv preprint arXiv:2406.15927*, 2024.
- [54] Chao Chen, Kai Liu, Ze Chen, Yi Gu, Yue Wu, Mingyuan Tao, Zhihang Fu, and Jieping Ye. INSIDE: llms’ internal states retain the power of hallucination detection. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.
- [55] Seongheon Park, Xuefeng Du, Min-Hsuan Yeh, Haobo Wang, and Yixuan Li. How to steer llm latents for hallucination detection? *arXiv preprint arXiv:2503.01917*, 2025.
- [56] Anastasios N Angelopoulos and Stephen Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511*, 2021.
- [57] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

- [58] Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*, volume 29. Springer, 2005.
- [59] Chen Xu and Yao Xie. Conformal prediction for time series. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(10):11575–11587, 2023. doi: 10.1109/TPAMI.2023.3272339.
- [60] Chen Xu and Yao Xie. Sequential predictive conformal inference for time series. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 38707–38727. PMLR, 23–29 Jul 2023. URL <https://proceedings.mlr.press/v202/xu23r.html>.
- [61] Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. In *8th Annual Conference on Robot Learning*, 2024.
- [62] Allen Z. Ren. open-pi-zero: Re-implementation of pi0 vision-language-action (vla) model from physical intelligence. <https://github.com/allenzren/open-pi-zero>, 2025. Version 0.1.1, released January 27, 2025. Accessed April 6, 2025.
- [63] Yashvir S Grewal, Edwin V Bonilla, and Thang D Bui. Improving uncertainty quantification in large language models via semantic embeddings. *arXiv preprint arXiv:2410.22685*, 2024.
- [64] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL <http://github.com/jax-ml/jax>.
- [65] Mingjian Jiang, Yangjun Ruan, Prasanna Sattigeri, Salim Roukos, and Tatsunori Hashimoto. Graph-based uncertainty metrics for long-form language model outputs. *arXiv preprint arXiv:2410.20783*, 2024.
- [66] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al.  $\pi_{0.5}$ : a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.
- [67] Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr0ot n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.
- [68] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023.
- [69] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.
- [70] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.
- [71] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

# Appendix

## A Experiment Details

### A.1 Vision-Language-Action Models

We conduct experiments on 3 state-of-the-art large VLA models: OpenVLA [2],  $\pi_0$  [4] and  $\pi_0$ -FAST [5]. Given the internal feature vectors  $E \in \mathbb{R}^{n \times d'}$  produced by a VLA model, where dimension  $n$  corresponds to different token positions, diffusion steps, etc. and  $d$  is the feature dimension, we aggregate  $E$  into a single fixed-dimensional feature vector  $e \in \mathbb{R}^d$  before inputting to the proposed *SAFE* models. In this paper, we consider and ablate the following ways of feature aggregation:

- First: take the first vector along the dimension  $n$ ,  $e = E_1$ ;
- Last: take the last vector along the dimension  $n$ ,  $e = E_n$ ;
- Mean: take average over the dimension  $n$ ,  $e = \frac{1}{n} \sum_{i=1}^n E_i$ ;
- First&Last: concatenate the first and the last vector,  $e = \text{concat}(E_0, E_n) \in \mathbb{R}^{2d'}$ .

Both OpenVLA and  $\pi_0$ -FAST first predict a sequence of discrete tokens and then convert them into continuous actions. We take feature vectors before being decoded into the output tokens from the last transformer block as  $E \in \mathbb{R}^{n \times d'}$ , and therefore  $n$  corresponds to the number of generated tokens. We ablate the aggregation method along this dimension and denote the aggregation method as  $\text{agg}_{\text{token}}$ . For  $\pi_0$ -FAST, we additionally ablate using the feature vectors before (“encoded”) and after (“pre-logits”) the final RMS normalization layer as  $E$ .

Differently,  $\pi_0$  (and  $\pi_0^*$ ) outputs action vectors by flow matching [68], and we take the feature vectors before being projected into the velocity field. Suppose  $\pi_0$  predict an action chunk of horizon  $H$  and performs  $k$  flow matching steps, the internal features become  $E \in \mathbb{R}^{H \times k \times d}$  and we perform the aggregation process along the  $H$  dimension and the  $k$  dimension separately to get the final embedding vector  $e \in \mathbb{R}^{d'}$ . The aggregation methods are denoted as  $\text{agg}_{\text{horiz}}$  and  $\text{agg}_{\text{diff}}$  along these two dimensions, respectively.

For all VLA models, we ablate different methods of aggregating the hidden features  $E$  into a single feature vector  $e$  and select the best method according to  $\mathcal{D}_{\text{eval-seen}}$  performance. The detailed ablation results are shown in Appendix A.7.

OpenVLA and  $\pi_0^*$  use MIT License;  $\pi_0$  and  $\pi_0$ -FAST use Apache-2.0 license.

### A.2 SAFE Failure Detector

*SAFE*-LSTM uses an LSTM model with 1 layer and a hidden dimension of 256, and an additional linear layer is used to project the hidden states of LSTM into a single scalar  $s_t$ . *SAFE*-MLP uses a multi-layer perceptron with 2 layers and a hidden dimension of 256. Since successes and failures from the generated rollouts are imbalanced, the losses on positive (failed) and negative (successful) rollouts are weighted by their inverse class frequency. We also apply an L2 regularization loss on the model weights to reduce overfitting, and this loss is weighted by  $\lambda_{\text{reg}}$  and optimized together with the failure score learning loss  $L_{\text{LSTM}}$  or  $L_{\text{MLP}}$ .  $\lambda_{\text{reg}}$  are determined by grid search.

### A.3 Failure Detection Baselines

For the cluster entropy baseline, we use agglomerative clustering with the ward linkage criterion [69]. The distance threshold is denoted as  $\delta$  and decided by grid search.

For the STAC baseline [17], we use the Maximum Mean Discrepancy (MMD) distance measure [70] with radial basis function kernels, which was reported to have the best performance by [17]. The bandwidth of the RBF kernel is 1.

For all baselines except for RND [39] and LogpZO [8], we ablate one version that only considers the failure score computed from the current timestep (“cumsum=False”) and another that uses the cumulative sum (cumsum) of the failure scores over time (“cumsum=True”).

For RND and LogpZO, we use the original implementation provided by the authors<sup>3</sup> and do not accumulate scores. In [8], RND and LogpZO are trained to model the distribution of (encoded) observations  $\mathbf{o}_t$  and predicted actions  $\mathbf{A}_t$ . In this work, we adapt them to model the distribution of VLA’s internal embeddings  $\mathbf{e}_t$ .

Note that as  $\pi_0$  (and  $\pi_0^*$ ) does not output discrete tokens, token uncertainty-based baselines do not apply. And for OpenVLA,  $H = H' = 1$  and thus the STAC [17] and STAC-Single do not apply.

#### A.4 Conformal Prediction

We follow [22, 8] for CP band construction. Please refer to Section. B in the Appendix of [8] for a detailed formulation. Specifically, in our experiments, we use the adaptive modulation function (Equation 2 in the Appendix of [8]), which models the non-extreme behaviors of the functional data.

#### A.5 Benchmark Details

**LIBERO** [20]: We adopt the LIBERO-10 task suite, which contains the most diverse set of objects, environments, and instructions among the 4 LIBERO task suites, and therefore LIBERO-10 is considered the most challenging task suite. LIBERO-10 contains 10 tasks with 50 rollouts in each task. We use the initial conditions for all rollouts as specified and provided by the author<sup>4</sup>. To test VLA models on LIBERO, we adopt the trained model weights provided by the respective authors and do not further finetune them. On LIBERO-10, OpenVLA achieves a success rate of 53.7%,  $\pi_0$ -FAST achieves 60.2%, and  $\pi_0$  achieves 85.2%. For evaluation, 3 out of 10 tasks are unseen, and within seen tasks, 60% of rollouts are used for  $\mathcal{D}_{\text{train}}$  and the remaining 40% for  $\mathcal{D}_{\text{eval-seen}}$ .

Note that the LIBERO simulator stops the rollout execution when the robot finishes the task (considered a success) or a maximum rollout length is reached (considered a failure). Therefore, in the generated rollouts, failed ones always have the maximum length, but successful ones are shorter. This could result in an unfair advantage for some of the compared failure detectors (if a failure detector simply learns to count the time elapsed, i.e.,  $s_t = t$ , it will achieve perfect failure detection since failed rollouts have a fixed and longer duration). To ensure a fair comparison, for evaluation in Table 1, we compute the minimum rollout length for each task and use that as  $T$  for that task. The failure detection performance (in ROC-AUC) is then determined based on  $s_T$ , where  $T$  is the same for all successful and failed rollouts within each task.

LIBERO benchmark uses the MIT license.

**SimplerEnv** [61]: SimplerEnv carefully identifies and reduces the domain gap between the simulation and the real-world demonstration data, and provides simulated environments that highly resemble the demonstration data from RT-series [1, 7, 32] (with the Google Robot embodiment) and BridgeData V2 [34] (with the WidowX embodiment). They show that models pretrained on real-world datasets can also accomplish similar tasks in SimplerEnv without finetuning, and their performance in simulation matches that in the real world.

On this benchmark, we adopt the pretrained model checkpoints of  $\pi_0^*$  [62]. Note that  $\pi_0^*$  model checkpoints are trained separately on the Google Robot embodiment and the WidowX embodiment, which results in two model checkpoints that have different internal feature spaces. Therefore, all failure detectors are trained and evaluated on each embodiment separately as well. All reported evaluation metrics are computed separately for each embodiment and then averaged. In Table 2, we list the tasks used for failure detection on SimplerEnv. We generate 100 rollouts for each task with random initial configurations, and the success rates of  $\pi_0^*$  on each task are also listed in Table 2. A rollout stops after the maximum number of allowed timesteps have passed, regardless of task success or failure. Within each embodiment, 1 out of 4 tasks is unseen, and within the seen tasks, 66% of the rollouts are in  $\mathcal{D}_{\text{train}}$  and the remaining 33% in  $\mathcal{D}_{\text{eval-seen}}$ .

SimplerEnv benchmark uses the MIT license.

**Real-world experiments with Franka robot:** In Table 3, we list the tasks used in the real-world experiments. For each task, we set a number of timesteps  $T$  allowed for one rollout, and all rollouts of

---

<sup>3</sup><https://github.com/CXU-TRI/FAIL-Detect>

<sup>4</sup>[https://github.com/Lifelong-Robot-Learning/LIBERO/tree/master/libero/libero/init\\_files](https://github.com/Lifelong-Robot-Learning/LIBERO/tree/master/libero/libero/init_files)

Table 2: List of tasks used in SimplerEnv benchmark.

Embodiment	Task ID	Environment Name	$\pi_0^*$ Success Rate (%)
Google Robot	1	google_robot_move_near_v0	77
Google Robot	2	google_robot_open_drawer	50
Google Robot	3	google_robot_close_drawer	80
Google Robot	4	google_robot_place_apple_in_closed_top_drawer	40
WidowX	1	widowx_carrot_on_plate	44
WidowX	2	widowx_put_eggplant_in_basket	88
WidowX	3	widowx_spoon_on_towel	79
WidowX	4	widowx_stack_cube	43

Table 3: List of tasks used in the real-world experiments.

Task	Instruction	Rollout Length $T$
1	close the door	300
2	close the drawer	200
3	pick up the ball and place it in the bowl	400
4	pick up the knife and put it on the plate	350
5	pick up the lid and place it on the pot	400
6	pick up the lid from the pot and place it on the table	400
7	pick up the marker and place it in the cup	400
8	place the green block on the yellow block	350
9	place the pink cup to the right of the blue cup	300
10	press the button	200
11	put both the carrot and the ball in the bowl	500
12	put the cup to the upright position	500
13	unfold the cloth	500

the same task are terminated after the same  $T$  timesteps regardless of task success or failure. In Fig. 6, we further visualize some example successful and failed rollouts from the real-world experiments.

### A.6 Training Details

We use Adam optimizer [71] with  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ ,  $\epsilon = 10^{-8}$ , and a learning rate (lr) determined by grid search. The *SAFE* models are trained for 1000 epochs with batch size 512. Note that each rollout is considered as one data point and thus batch size of 512 translates to training on (at most) 512 rollouts in each iteration. All training and evaluation are done on a single NVIDIA A100 40GB GPU. Since *SAFE* uses small networks (MLP or LSTM with 1 or 2 layers), the typical training time for one model is less than one minute.

### A.7 Hyperparameter Tuning

To determine the hyperparameters for the proposed *SAFE* and baselines, we perform a grid search over them and select the ones with the highest failure detection performance (ROC-AUC) on the  $\mathcal{D}_{\text{eval-seen}}$  split. In Table 5, Table 6, and Table 7, we report the hyperparameters we have searched over and the values with the best performance. Note that for the real-world experiments, we fix the  $\mathbf{e}_t$  to be the ‘‘pre-logits’’ with ‘‘Mean’’ aggregation.

## B Additional Results and Discussions

### B.1 Feature Visualization and Analysis

We perform the feature analysis similar to Section 4.1 and Fig. 1 on other benchmarks and show the plots in Fig. 7. Note that in this feature analysis process, the t-SNE algorithm was performed on the VLA’s embeddings without any learning. Therefore, the feature dimension reduction process is unsupervised and does not know about task successes or failures.

Comparing the plots in Fig. 1 and Fig. 7, we can see that the embedding spaces from VLAs are different from each other, which corresponds to the different failure patterns presented by the VLAs. For  $\pi_0$ -FAST (Fig. 1) and  $\pi_0$  on LIBERO (Fig. 7a and b), when task execution fails, the embeddings fall into the same region (“failure zone”). This corresponds to the major failure mode of  $\pi_0$ -FAST trained on the LIBERO dataset, where the predicted actions  $A_t$  become unstable and the robot arms move to weird configurations and out of the observation frame. For OpenVLA on LIBERO (Fig. 7c and d), we observe that for most failed rollouts, the robot freezes at or shakes around certain configurations during the middle of task execution. Such failed rollouts result in features very close to each other, which corresponds to small blobs of red dots in Fig. 7c.

Despite the different appearances of the embedding spaces from the above benchmarks, their successful and failed rollouts are separable in the feature space. This is aligned with the high performance of the proposed *SAFE* and the embedding-based baseline methods. Moreover, although the embeddings of the failed rollouts from OpenVLA are spread over the space and do not form a unified “failure zone”, *SAFE* is still able to learn to separate task failures from successes (possibly by extracting the correlations that are not visualized by t-SNE) and generalize well to unseen tasks, as reported in Table 1.

However, the visualized embeddings of  $\pi_0$ -FAST on the real Franka robot (Fig. 7e and f) are different, where embeddings from successful and failed rollouts are not easily separable through the t-SNE visualization. We hypothesize that because the tasks we used for real-world experiments are more diverse, their failures do not have a unified semantic meaning, and thus the embeddings are not clearly separated in the visualization. This explains the limited performance of all failure detection methods as reported by Fig. 5, where ROC-AUC is at most 64 on  $\mathcal{D}_{\text{eval-unseen}}$ . Nevertheless, *SAFE*-MLP still outperforms all baselines on both seen and unseen splits in this evaluation.

## B.2 Conformal Prediction Results

We use functional CP [8, 22] to determine the time-varying thresholds  $\delta_t$  for failure detection. By varying the significance level  $\alpha$  used in functional CP, we can adjust the conservativeness of failure detection and get different performance. In Fig. 8, we plot the change of TNR (True Negative Rate), TPR (True Positive Rate) and Bal-acc (Balanced Accuracy,  $\frac{\text{TNR}+\text{TPR}}{2}$ ) w.r.t.  $\alpha$ .

Note that we calibrate the CP bands on successful rollouts (negative data points), and thus if the assumptions used in CP ( $s_t$  are sampled i.i.d.) hold, the TNR rate is lower bounded by and close to  $1 - \alpha$  (the gray dashed line in the TNR plots in Fig. 8) [22]. However, as the multitask failure detection problem requires detecting failures on tasks that are not in the training or the calibration sets, we need to calibrate CP bands on  $\mathcal{D}_{\text{eval-seen}}$  and then evaluate them on  $\mathcal{D}_{\text{eval-unseen}}$ . Therefore, the i.i.d assumption may not hold, and TNR may deviate from the gray dashed line.

From Fig. 8, we can see that on OpenVLA+LIBERO and  $\pi^*$ +SimplerEnv benchmarks, the TNR curves obtained by *SAFE* are close to the gray dashed line  $1 - \alpha$ , while those on the other 3 benchmarks are lower than  $1 - \alpha$ . A similar phenomenon is also observed for the baseline methods: none of the TNR curves obtained from the baselines consistently conform to the  $1 - \alpha$  curve across all benchmarks. We attribute this to the challenging nature of the multitask failure detection problem, where the failure scores for calibration and evaluation may not come from the same distribution. Nevertheless, we still adopt the functional CP as a principled method to determine the time-varying failure detection threshold  $\delta_t$ . Moreover, from Fig. 8, we can see that *SAFE* can achieve higher TPR and result in fewer false negatives compared to the baselines. This is crucial for safety-critical environments, where a missing failure (false negative) can be much more catastrophic than a false alarm (false positive).

## B.3 Failure Detection Time

As mentioned in Section 6.2, we manually label when the failure happens for the failed rollouts. The labeling process is based on video recordings after all rollouts are collected and no interventions were done during the task execution. While the exact times of failure are clear for some failure modes (e.g. dangerous actions, breaking objects), they can be ambiguous and hard to annotate for other failure modes. For example, a policy may freeze in the middle of task execution, and after that either recovering from it or getting stuck indefinitely can be possible. In another case, a policy may repeatedly try grasping the object but keep missing the grasp until timeout, and it’s hard to determine

Table 4: Mean and standard deviation of failure detection ROC-AUC on all benchmarks. This table complements the results from Table 1 and Fig. 5 left.

VLA Model Benchmark Eval Task Split	OpenVLA LIBERO		$\pi_0$ -FAST LIBERO		$\pi_0$ LIBERO		$\pi_0^*$ SimplerEnv		$\pi_0$ -FAST Real Franka	
	Seen	Unseen	Seen	Unseen	Seen	Unseen	Seen	Unseen	Seen	Unseen
Max prob.	50.25 $\pm$ 2.51	53.83 $\pm$ 6.32	61.32 $\pm$ 9.57	69.44 $\pm$ 13.61	-	-	-	-	53.74 $\pm$ 3.46	48.59 $\pm$ 3.00
Avg prob.	44.05 $\pm$ 1.26	51.58 $\pm$ 1.82	52.46 $\pm$ 3.44	58.04 $\pm$ 5.64	-	-	-	-	51.60 $\pm$ 3.12	47.30 $\pm$ 4.32
Max entropy	52.94 $\pm$ 4.36	53.09 $\pm$ 7.68	46.69 $\pm$ 13.33	62.96 $\pm$ 19.62	-	-	-	-	59.23 $\pm$ 3.06	53.50 $\pm$ 3.15
Avg entropy	45.27 $\pm$ 1.78	50.03 $\pm$ 3.18	50.93 $\pm$ 1.22	58.63 $\pm$ 3.47	-	-	-	-	50.67 $\pm$ 3.96	46.08 $\pm$ 4.79
Mahalanobis dist.	62.03 $\pm$ 5.11	58.85 $\pm$ 4.16	93.56 $\pm$ 2.32	83.79 $\pm$ 7.18	77.12 $\pm$ 8.57	74.31 $\pm$ 12.64	88.42 $\pm$ 2.82	52.84 $\pm$ 31.97	75.54 $\pm$ 4.07	53.93 $\pm$ 5.06
Euclidean dist. $k$ -NN	66.00 $\pm$ 2.33	55.23 $\pm$ 10.05	92.04 $\pm$ 2.39	84.12 $\pm$ 6.47	75.64 $\pm$ 6.20	70.73 $\pm$ 16.69	89.73 $\pm$ 3.08	68.41 $\pm$ 9.22	80.35 $\pm$ 5.36	60.27 $\pm$ 4.79
Cosine dist. $k$ -NN	67.09 $\pm$ 2.74	69.45 $\pm$ 6.14	92.09 $\pm$ 1.70	84.64 $\pm$ 4.90	75.76 $\pm$ 6.16	70.31 $\pm$ 16.84	90.19 $\pm$ 4.05	71.32 $\pm$ 12.02	80.23 $\pm$ 5.12	59.51 $\pm$ 5.76
PCA-KMeans [9]	57.18 $\pm$ 2.04	55.10 $\pm$ 1.16	68.46 $\pm$ 4.92	57.12 $\pm$ 10.44	64.92 $\pm$ 8.90	60.35 $\pm$ 19.99	66.88 $\pm$ 5.10	61.19 $\pm$ 14.76	51.91 $\pm$ 4.20	49.86 $\pm$ 6.19
RND [39]	52.57 $\pm$ 4.56	46.88 $\pm$ 4.92	88.67 $\pm$ 3.05	81.57 $\pm$ 8.67	71.92 $\pm$ 7.02	69.44 $\pm$ 19.30	85.07 $\pm$ 4.00	65.89 $\pm$ 6.52	62.00 $\pm$ 5.44	45.83 $\pm$ 5.10
LogpZO [8]	61.57 $\pm$ 3.62	52.91 $\pm$ 5.79	91.52 $\pm$ 2.39	83.07 $\pm$ 7.17	76.80 $\pm$ 9.12	73.23 $\pm$ 11.64	88.79 $\pm$ 4.92	74.66 $\pm$ 14.96	64.43 $\pm$ 7.82	52.24 $\pm$ 3.68
Action total var.	62.76 $\pm$ 1.66	65.43 $\pm$ 2.50	76.95 $\pm$ 7.22	74.50 $\pm$ 12.19	77.20 $\pm$ 5.65	75.18 $\pm$ 5.08	68.41 $\pm$ 10.81	67.94 $\pm$ 15.97	-	-
Trans. total var.	55.33 $\pm$ 2.06	58.99 $\pm$ 5.13	78.21 $\pm$ 4.09	80.03 $\pm$ 9.11	49.38 $\pm$ 9.95	54.71 $\pm$ 7.57	63.27 $\pm$ 7.57	55.90 $\pm$ 19.19	-	-
Rot. total var.	47.85 $\pm$ 2.88	55.30 $\pm$ 4.38	80.87 $\pm$ 5.85	77.29 $\pm$ 8.71	52.94 $\pm$ 7.56	61.06 $\pm$ 10.69	58.07 $\pm$ 10.41	62.10 $\pm$ 9.39	-	-
Gripper total var.	61.84 $\pm$ 2.67	64.48 $\pm$ 3.05	76.82 $\pm$ 7.10	74.42 $\pm$ 12.13	77.19 $\pm$ 5.66	75.19 $\pm$ 5.08	69.16 $\pm$ 9.50	69.29 $\pm$ 14.77	-	-
Cluster entropy	50.16 $\pm$ 2.36	51.44 $\pm$ 1.01	80.22 $\pm$ 7.37	80.53 $\pm$ 8.65	76.19 $\pm$ 4.31	72.12 $\pm$ 1.04	68.25 $\pm$ 9.03	73.66 $\pm$ 16.03	76.44 $\pm$ 9.03	-
STAC [17]	-	-	83.07 $\pm$ 4.61	85.31 $\pm$ 6.71	46.55 $\pm$ 8.90	47.91 $\pm$ 20.94	60.74 $\pm$ 13.89	62.21 $\pm$ 16.72	-	-
STAC-Single	-	-	85.46 $\pm$ 6.55	81.16 $\pm$ 8.63	68.46 $\pm$ 5.10	69.39 $\pm$ 8.22	68.71 $\pm$ 7.00	70.40 $\pm$ 8.76	45.24 $\pm$ 3.68	38.01 $\pm$ 9.81
SAFE-LSTM	70.24 $\pm$ 1.49	72.47 $\pm$ 5.55	92.98 $\pm$ 2.62	84.48 $\pm$ 7.29	76.98 $\pm$ 5.34	71.09 $\pm$ 6.94	88.85 $\pm$ 6.30	80.11 $\pm$ 10.49	77.27 $\pm$ 5.82	58.70 $\pm$ 4.37
SAFE-MLP	72.68 $\pm$ 2.38	73.47 $\pm$ 3.39	90.06 $\pm$ 2.82	80.44 $\pm$ 5.72	73.50 $\pm$ 7.43	73.27 $\pm$ 11.88	89.50 $\pm$ 4.49	84.82 $\pm$ 8.12	86.76 $\pm$ 2.64	64.16 $\pm$ 5.88

a single point of failure. To handle such cases, we instruct the human annotators to pick the time where they think intervention is needed and they should take over control to prevent an execution failure. In practice, for the above ambiguous failure modes, we annotate the failures after the policy gets stuck by a few seconds or re-tries the grasping action a few times. For some rollouts that look very plausible but do not succeed due to the time limit, the failure time is annotated as the end of the rollout. Note that we annotate only the failed rollouts and not the successful ones, even though they may also show subtle signs of failure in the middle.

In Fig. 9, we compare the times of failure detected by the proposed SAFE-MLP model and a human annotator. From Fig. 9, we can see that for both  $\pi_0$  and  $\pi_0$ -FAST models, SAFE-MLP can detect failures before they happen (as identified by a human). When used for  $\pi_0$ -FAST deployed on LIBERO, SAFE-MLP can *forecast* failures well in advance and even predict 40% of the failures after the first timestep.

Furthermore, from Fig. 9a and Fig. 9c, we can see that the blue curves jump up on the right edge of the plots. This means that the human annotator does not think these rollouts are failures until the very last moment, where the VLA model is probably on the right track and fails only due to timeout. We think such failures are also hard for failure detectors to detect, and it explains the low performance of all failure detectors on these benchmarks.

#### B.4 Result Variance

In Table 4, we report the standard deviation for all results in Table 1 and Fig. 5 left. Note that for the repeated runs, not only are they using different random seeds, also the tasks are split differently into the seen and the unseen subsets. Since different tasks have different difficulties for failure detection, it is normal to see large standard deviations in Table 4. From Table 4, we can see that the proposed SAFE methods achieve high averaged performance with relatively low standard deviations compared to the baselines, across all evaluation benchmarks. This signifies the strong and also stable performance of SAFE.

Table 5: Grid-searched and best-performing hyperparameters (in bold text) for OpenVLA+LIBERO (left) and  $\pi_0$ -FAST+LIBERO (right).

Method	HParams	Values	Method	HParams	Values
Max prob.	cumsum	True <b>False</b>	Mahalanobis dist.	cumsum	True <b>False</b>
Avg prob.	cumsum	True <b>False</b>		cumsum	True <b>False</b>
Max entropy	cumsum	True <b>False</b>		cumsum	True <b>False</b>
Avg entropy	cumsum	True <b>False</b>		cumsum	True <b>False</b>
Mahalanobis dist.	<i>agg</i> <sub>token</sub>	First Last Mean	<i>agg</i> <sub>token</sub>	First Last <b>Mean</b>	First Last <b>Mean</b>
	cumsum	True <b>False</b>		Feat	Encoded <b>Pre-logits</b>
Euclidean dist. <i>k</i> NN	<i>agg</i> <sub>token</sub>	First Last <b>Mean</b>	cumsum	True <b>False</b>	True <b>False</b>
	cumsum	<b>True False</b>	<i>agg</i> <sub>token</sub>	First Last Mean	First Last Mean
Cosine dist. <i>k</i> NN	<i>agg</i> <sub>token</sub>	First Last Mean	Feat	Encoded <b>Pre-logits</b>	Encoded <b>Pre-logits</b>
	cumsum	<b>True False</b>	cumsum	True <b>False</b>	True <b>False</b>
PCA-KMeans	<i>agg</i> <sub>token</sub>	1 5 <b>10</b>	<i>agg</i> <sub>token</sub>	1 5 <b>10</b>	1 5 <b>10</b>
	cumsum	<b>True False</b>	Feat	First Last Mean	First Last Mean
RND	<i>agg</i> <sub>token</sub>	1 5 <b>10</b>	cumsum	Encoded <b>Pre-logits</b>	Encoded <b>Pre-logits</b>
LogpZO	<i>agg</i> <sub>token</sub>	First Last Mean	<i>agg</i> <sub>token</sub>	True False	True False
	<i>agg</i> <sub>token</sub>	<b>First Last Mean</b>	Feat	16 32 64	16 32 64
Action total var.	cumsum	True <b>False</b>	clusters	32 64 128	32 64 128
Trans. total var.	cumsum	True <b>False</b>	dim		
Rot. total var.	cumsum	<b>True False</b>	<i>agg</i> <sub>token</sub>	First Last Mean	First Last Mean
Gripper total var.	cumsum	<b>True False</b>	Feat	Encoded <b>Pre-logits</b>	Encoded <b>Pre-logits</b>
Cluster entropy	cumsum	<b>True False</b>	<i>agg</i> <sub>token</sub>	First Last Mean	First Last Mean
	$\delta$	0.01 0.05	Feat	Encoded <b>Pre-logits</b>	Encoded <b>Pre-logits</b>
<i>SAFE-LSTM</i>	<i>agg</i> <sub>token</sub>	First Last Mean	Action total var.	cumsum	True <b>False</b>
	lr	<b>1e-4</b> 3e-4 1e-3	Trans. total var.	cumsum	True <b>False</b>
<i>SAFE-MLP</i>	$\lambda_{reg}$	1e-3 1e-2 1e-1 <b>1</b>	Rot. total var.	cumsum	True <b>False</b>
	<i>agg</i> <sub>token</sub>	First Last Mean	Gripper total var.	cumsum	True <b>False</b>
	lr	<b>1e-4</b> 3e-4 1e-3	Cluster entropy	$\delta$	True <b>False</b>
	$\lambda_{reg}$	1e-3 <b>1e-2</b> 1e-1 1			0.01 0.05 0.1 <b>0.2</b> 0.5 1 2 5
STAC	cumsum	True <b>False</b>	<i>SAFE-LSTM</i>	<i>agg</i> <sub>token</sub>	First Last <b>Mean</b>
STAC-Single	cumsum	True <b>False</b>		Feat	Encoded <b>Pre-logits</b>
				lr	3e-5 1e-4 <b>3e-4</b> 1e-3
				$\lambda_{reg}$	<b>1e-3</b> 1e-2 1e-1
			<i>SAFE-MLP</i>	<i>agg</i> <sub>token</sub>	First Last Mean
				Feat	Encoded <b>Pre-logits</b>
				lr	3e-5 <b>1e-4</b> 3e-4 1e-3
				$\lambda_{reg}$	1e-3 <b>1e-2</b> 1e-1

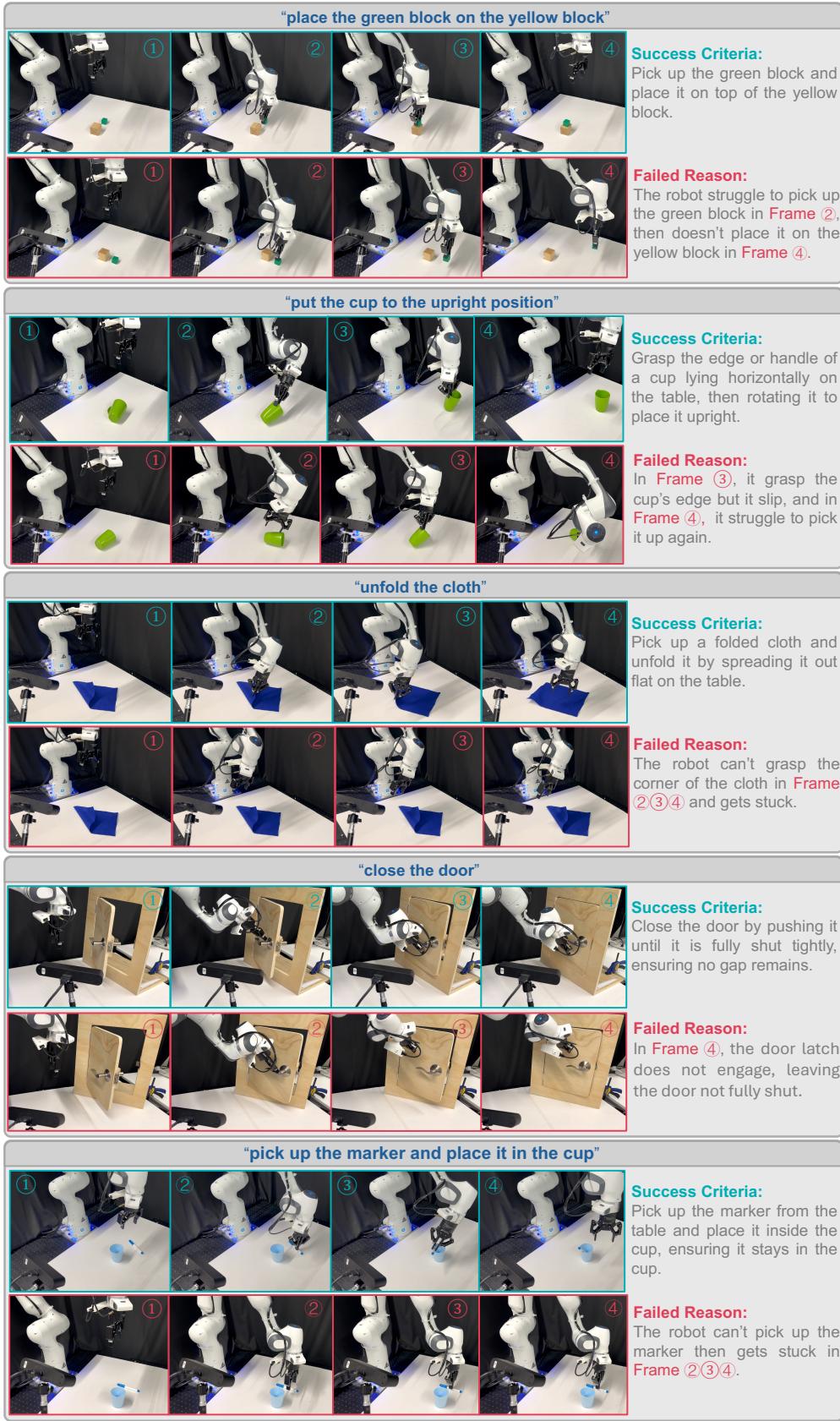
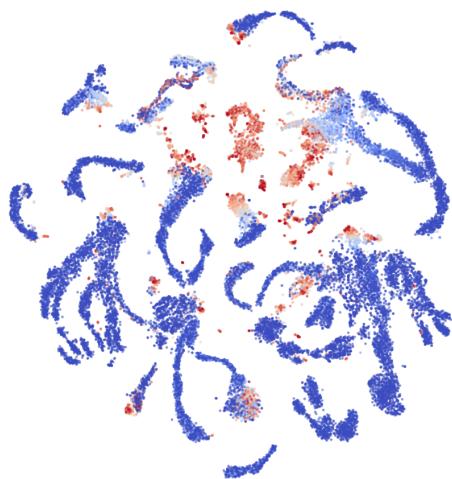


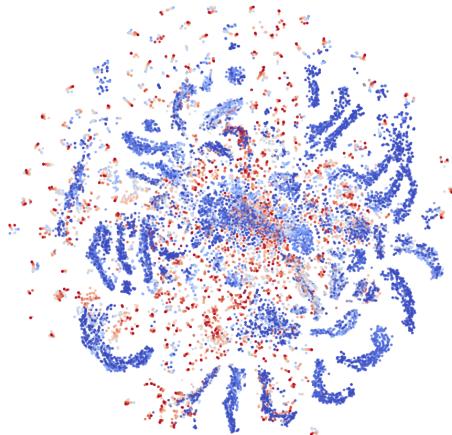
Figure 6: Example successful and failed rollouts from real-world experiments.



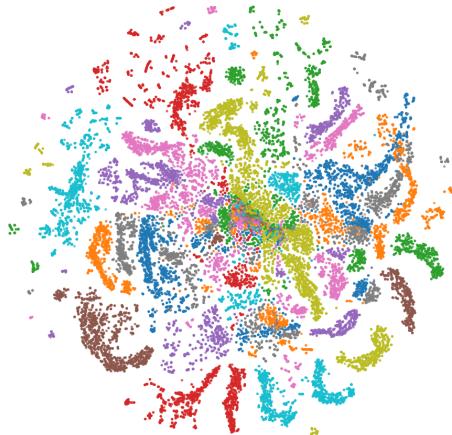
(a)  $\pi_0$ +LIBERO, colored by task success



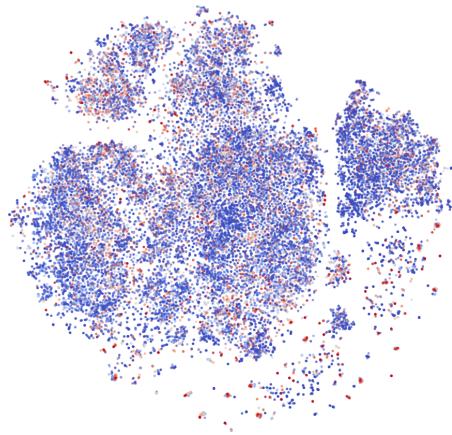
(b)  $\pi_0$ +LIBERO, colored by task ID



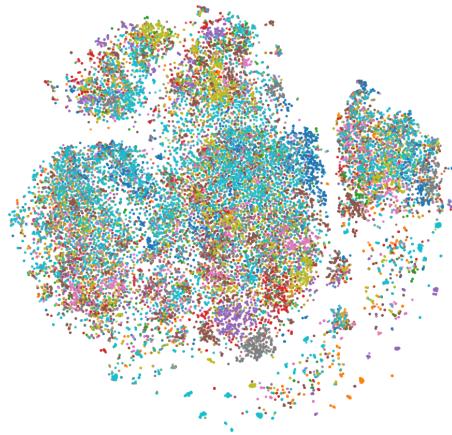
(c) OpenVLA+LIBERO, colored by task success



(d) OpenVLA+LIBERO, colored by task ID



(e)  $\pi_0$ -FAST+Franka, colored by task success



(f)  $\pi_0$ -FAST+Franka, colored by task ID

Figure 7: t-SNE plots of VLA's internal features, from different evaluation benchmarks.

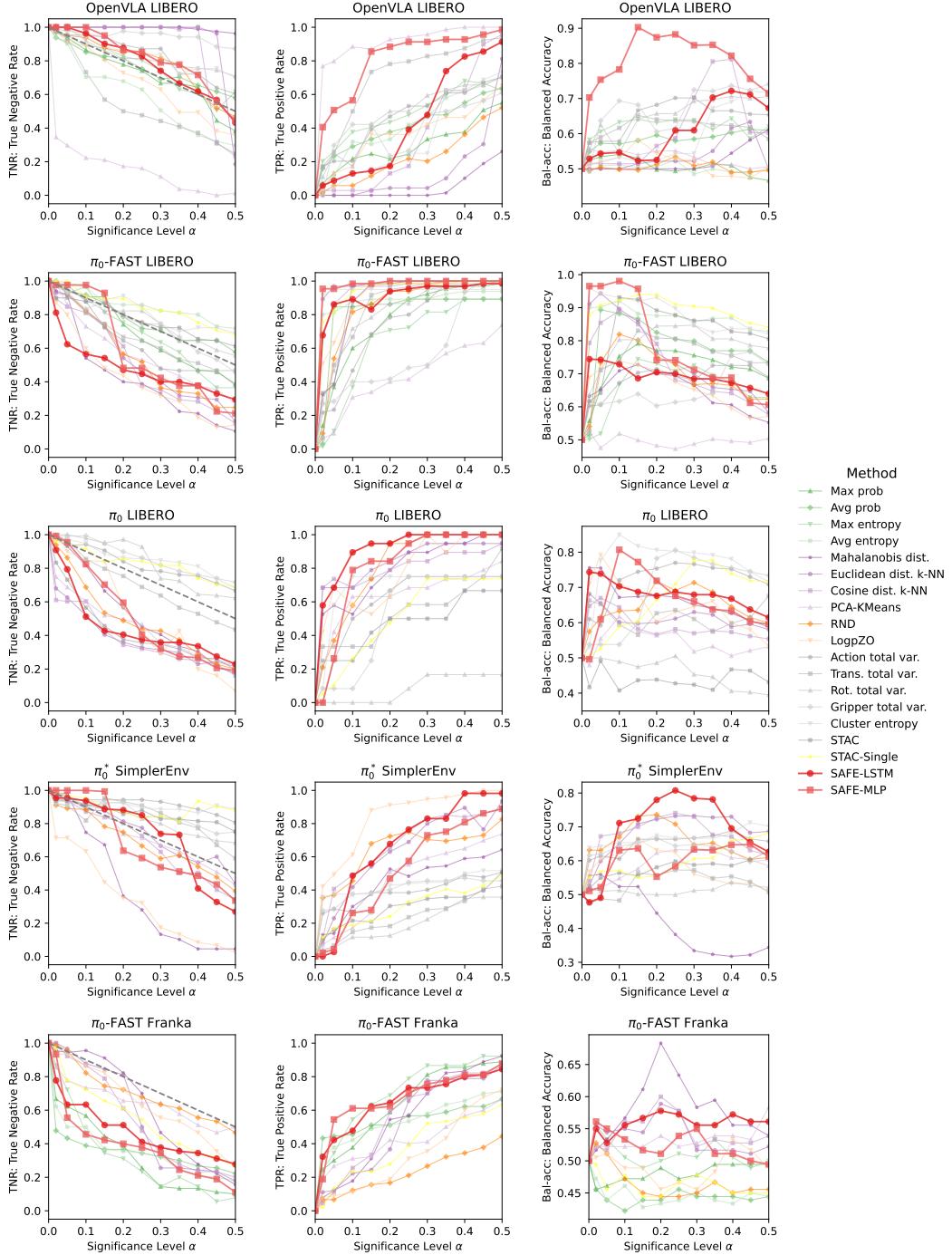


Figure 8: Additional failure detection results using  $\delta_t$  obtained by functional CP. These plots show TNR (left column), TPR (middle column), and Bal-acc (right column) w.r.t. the significance level  $\alpha$ , for each evaluation benchmark. These plots are obtained with random seed= 0.

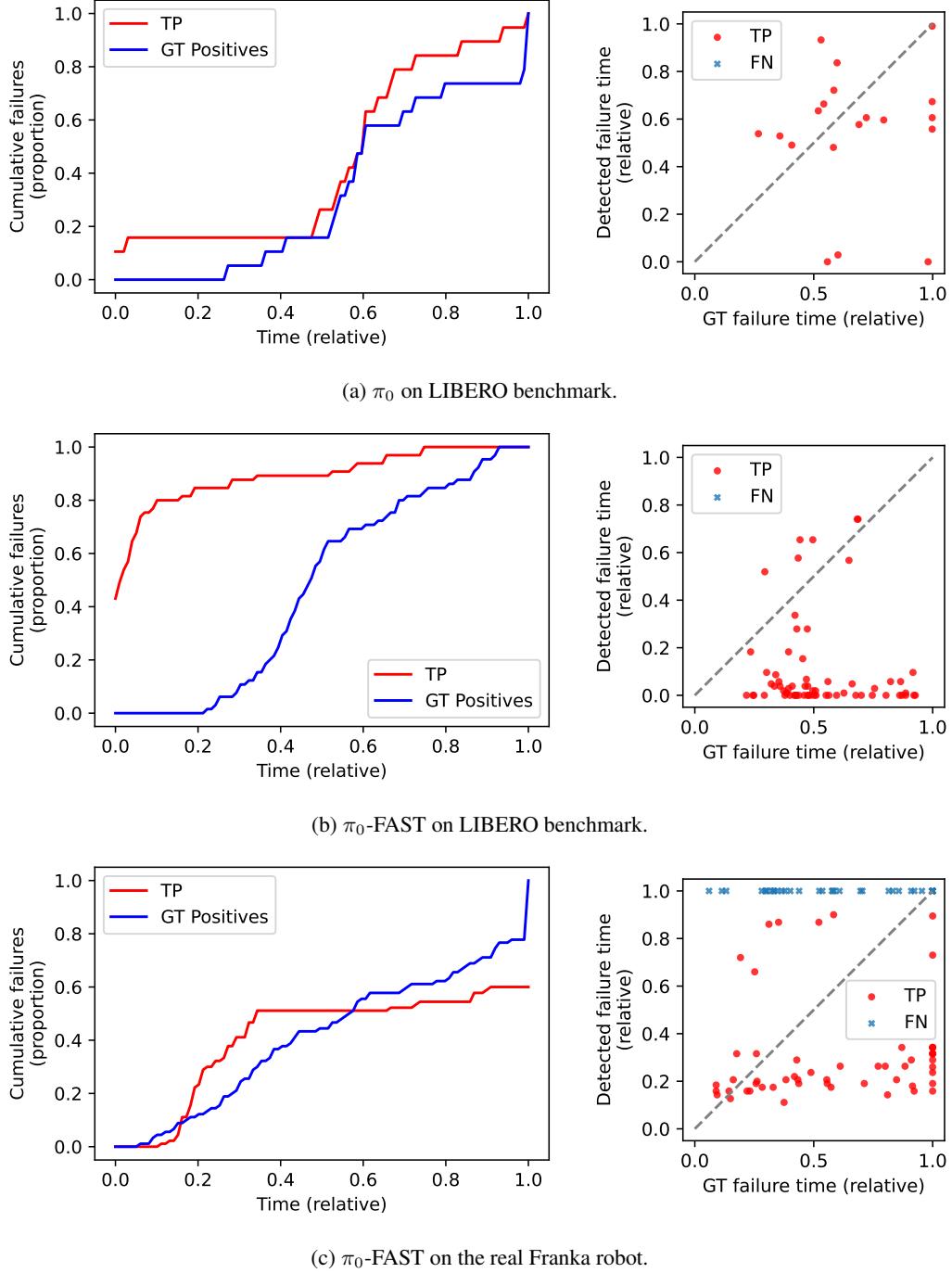


Figure 9: Comparison between detected and ground truth (GT) failure w.r.t time. On the left column, we plot the cumulative number of true failures (true positives) detected by *SAFE-MLP* (red) and a human annotator (blue), w.r.t. elapsed time in each rollout. The right column shows the time of failures detected by *SAFE-MLP* (y-axis) and a human annotator (x-axis) for each rollout, where failures missed by the detector (false negatives) are plotted in blue crosses. Experiments are done with seed 0 and functional CP with significance level  $\alpha = 0.15$ .

Table 6: Grid-searched and best-performing hyperparameters (in bold text) for  $\pi_0$ +LIBERO (left) and  $\pi_0^*$ +SimplerEnv (right).

Method	HParams	Values	Method	HParams	Values
Mahalanobis dist.	$agg_{hori}$ $agg_{diff}$ cumsum	<b>First</b> Last First&Last First <b>Last</b> First&Last True <b>False</b>	Mahalanobis dist.	$agg_{hori}$ $agg_{diff}$ cumsum	First Last <b>Mean</b> First&Last First <b>Last</b> Mean First&Last True <b>False</b>
Euclidean dist. $k$ -NN	$agg_{hori}$ $agg_{diff}$ cumsum	<b>First</b> Last First&Last First <b>Last</b> First&Last True <b>False</b>	Euclidean dist. $k$ -NN	$agg_{hori}$ $agg_{diff}$ cumsum	First Last <b>Mean</b> First&Last First Last <b>Mean</b> First&Last True <b>False</b>
Cosine dist. $k$ -NN	$agg_{hori}$ $agg_{diff}$ cumsum	<b>First</b> Last First&Last First <b>Last</b> First&Last True <b>False</b>	Cosine dist. $k$ -NN	$agg_{hori}$ $agg_{diff}$ cumsum	First Last <b>Mean</b> First&Last First Last <b>Mean</b> First&Last True <b>False</b>
PCA-KMeans	$agg_{hori}$ $agg_{diff}$ cumsum	<b>First</b> Last First&Last First <b>Last</b> First&Last True <b>False</b>	PCA-KMeans	$agg_{hori}$ $agg_{diff}$ cumsum	First Last <b>Mean</b> First&Last First Last <b>Mean</b> First&Last True <b>False</b>
RND	$agg_{hori}$ $agg_{diff}$	<b>First</b> Last First&Last First <b>Last</b> First&Last	RND	$agg_{hori}$ $agg_{diff}$	First Last <b>Mean</b> First&Last First Last <b>Mean</b> First&Last
LogpZO	$agg_{hori}$ $agg_{diff}$	First Last <b>First&amp;Last</b>	LogpZO	$agg_{hori}$ $agg_{diff}$	First Last <b>Mean</b> First&Last First Last <b>Mean</b> First&Last
Action total var.	cumsum	True <b>False</b>	Action total var.	cumsum	True <b>False</b>
Trans. total var.	cumsum	True <b>False</b>	Trans. total var.	cumsum	True <b>False</b>
Rot. total var.	cumsum	<b>True</b> False	Rot. total var.	cumsum	<b>True</b> False
Gripper total var.	cumsum	True <b>False</b>	Gripper total var.	cumsum	True <b>False</b>
Cluster entropy	cumsum	True <b>False</b>	Cluster entropy	$\delta$	0.01 0.05 0.1 0.2 0.5 <b>1</b> 2 5
STAC	cumsum	True <b>False</b>	STAC	cumsum	True <b>False</b>
STAC-Single	cumsum	True <b>False</b>	STAC-Single	cumsum	True <b>False</b>
SAFE-LSTM	$agg_{hori}$ $agg_{diff}$ lr	<b>First</b> Last First&Last First <b>Last</b> First&Last 1e-5 3e-5 1e-4 3e-4 <b>1e-3</b>	SAFE-LSTM	$agg_{hori}$ $agg_{diff}$ lr	First Last <b>Mean</b> First&Last First Last <b>Mean</b> <b>First&amp;Last</b> 1e-4 3e-4 <b>1e-3</b>
SAFE-MLP	$\lambda_{reg}$ $agg_{hori}$ $agg_{diff}$ lr	<b>1e-3</b> 1e-2 1e-1 <b>First</b> Last First&Last First <b>Last</b> First&Last 1e-5 <b>3e-5</b> 1e-4 3e-4 1e-3	SAFE-MLP	$\lambda_{reg}$ $agg_{hori}$ $agg_{diff}$ lr	1e-3 <b>1e-2</b> 1e-1 First Last <b>Mean</b> First&Last First Last <b>Mean</b> <b>First&amp;Last</b> 1e-4 <b>3e-4</b> 1e-3
	$\lambda_{reg}$	<b>1e-3</b> 1e-2 1e-1		$\lambda_{reg}$	<b>1e-3</b> 1e-2 1e-1

Table 7: Grid-searched and best-performing hyperparameters (in bold text) for  $\pi_0$ -FAST on real-world rollouts.

Method	HParams	Values
Max prob.	cumsum	True False
Avg prob.	cumsum	True False
Max entropy	cumsum	True False
Avg entropy	cumsum	True False
Mahalanobis dist.	cumsum	True False
Euclidean dist. $k$ -NN	cumsum	True False
	$k$	1 <b>5</b> 10
Cosine dist. $k$ -NN	cumsum	True False
	$k$	1 <b>5</b> 10
PCA-KMeans	cumsum	True False
	clusters	16 <b>32</b> 64
	dim	32 64 <b>128</b>
STAC-Single	cumsum	True False
SAFE-LSTM	lr	1e-4 3e-4 <b>1e-3</b> 3e-3
	$\lambda_{reg}$	1e-3 <b>1e-2</b> 1e-1
SAFE-MLP	lr	1e-4 <b>3e-4</b> 1e-3 3e-3
	$\lambda_{reg}$	<b>1e-3</b> 1e-2 1e-1