

Unpacking Failure Modes of Generative Policies: Runtime Monitoring of Consistency and Progress

Anonymous Author(s)

Affiliation

Address

email

Abstract: Robot behavior policies trained via imitation learning are prone to failure under conditions that deviate from their training data. Thus, algorithms that monitor learned policies at test time and provide early warnings of failure are necessary to facilitate scalable deployment. We propose a runtime monitoring framework that splits the detection of failures into two complementary categories: 1) Erratic failures, which we propose to detect using a statistical measure of temporal action consistency, and 2) task progression failures, where we use VLMs to detect when the policy confidently and consistently takes actions that do not solve the task. Our approach has two key strengths. First, because learned policies exhibit diverse failure modes, combining complementary detectors leads to significantly higher accuracy at failure detection. Second, using a statistical temporal action consistency measure ensures that we quickly detect when multi-modal, generative policies exhibit erratic behavior at negligible computation cost. In contrast, we only use VLMs to detect failure modes that are less time-sensitive. We demonstrate our approach in the context of diffusion policies trained on a multi-modal domain and simulated bi-manual robotic manipulation domains. The resultant system, unifying temporal consistency detection with VLM runtime monitoring, detects 15% more failures than using either of the two detectors, thus highlighting the importance of assigning specialized detectors to complementary categories of failure. **Qualitative results** are made available at sites.google.com/view/detecting-policy-failure.

Keywords: Failure Detection, Generative Policies, Vision-Language Models

1 Introduction

Imitation learning represents one of the simplest yet most effective ways of learning robotic control behaviors from demonstration. Herein, the use of generative modeling techniques has been instrumental in allowing robot policies to learn the multi-modal landscape of solutions demonstrated by humans [1, 2]. When deployed, however, such robot policies will inevitably encounter *out-of-distribution* (OOD) test cases—scenarios that differ from the training set—on which the policy’s behavior cannot be known beforehand [3]. Thus, we require methods that monitor the behavior of learned policies at deployment time to detect whether they are failing as a result of distribution shift.

Identifying when a learned model is performing unreliably is typically formulated as an OOD detection problem, for which a taxonomy of methods exist [4, 5]. However, the use of such methods for detecting robot policy failure is met promptly with several challenges. For example, methods that detect when the world state differs from those contained in the training dataset [6, 7] do not account cases in which the policy succeeds or *generalizes* to unseen states, while other methods may not be directly applicable [8] to generative policy formulations; e.g., a diffusion policy (DP) which produces action sequences via iterative denoising [1]. The special case of multi-modal, generative robot policies necessitates the design of new failure detectors that can recognize their diverse failure modes beyond the training distribution.

The **key insight** of this paper is that we can split the task of detecting policy failure into two complementary categories, each of which can be addressed with a failure detector suited to the requirements of

40 their assigned category. The first is the detection of failures in which the policy exhibits erratic behavior
 41 as measured by the temporal inconsistency of its predicted actions over time. For example, if the policy
 42 repeatedly alternates between different action modes, causing the robot to bump into its surroundings.
 43 The second category is the detection of failures in which the policy is temporally consistent but struggles
 44 to make progress on its task, thereby requiring visually-grounded reasoning over longer durations of
 45 time by Vision-Language Models (VLMs). For example, the robot can stall in place or drift astray if the
 46 policy produces constant outputs. Notably, one would want to catch erratic failures (the first category)
 47 fast, whereas task progression failures (the second category) do not require immediate intervention.
 48 However, using naive temporal consistency checks on actions (e.g., the agreement of individual
 49 actions sampled from a DP over time) to detect erratic policy behavior is insufficient, simply because
 50 sampled actions are expected to change across inference steps of a generative policy. Thus, we propose
 51 to characterize *how much* a generative policy’s action distributions are changing across time using
 52 Statistical measures of Temporal Action Consistency (STAC). We propose to detect task progression
 53 failures (undetectable by STAC) zero-shot with a VLM, which can distinguish off-nominal behavior
 54 when prompted to reason about the robot’s progress in a video question answering setup.
 55 To summarize, our contributions are three-fold: 1) A formulation of failure detection for generative
 56 policies that splits failures into two complementary categories, thus admitting the use of specialized
 57 detectors toward system-level performance increases; 2) We propose STAC, a novel temporal
 58 consistency detector that measures the statistical consistency of a generative policy’s action
 59 distributions to detect erratic failures; 3) We propose the use of VLMs to monitor the task progress of a
 60 policy over the duration of its rollout, and we provide practical insights for their use as failure detectors.
 61 The combined system, which integrates STAC and the VLM runtime monitor, detects 90% of failures
 62 exhibited by diffusion policies across two simulated bi-manual robotic manipulation domains.

63 2 Related Work

64 Recent advances in **robotic imitation learning** include new policy architectures [9, 10, 11], hardware in-
 65 novations for data collection [12, 13, 14], community-wide efforts to scale robotic learning datasets [15,
 66 16, 17], and training high-capacity policies on these datasets [18, 19, 20]. Of recent interest is the use
 67 of generative models [21, 22, 23] to represent policies due to the inherent multi-modality contained in
 68 imitation learning datasets collected by humans; a setting in which generative models thrive. Diffusion
 69 policies [1], for example, formulate action sequence prediction as an iterative denoising process starting
 70 from sampled noise. In this work, we focus on characterizing the behavior of generative robot policies.
 71 Despite these advances, it is well known that **learned policies may fail** beyond their training
 72 distribution [3, 5, 4], in part due to compounding prediction errors on states induced by the
 73 policy [24, 25]. Thus, a line of works propose to retrain or adapt the policy on out-of-distribution states
 74 using corrective supervision from humans [26, 27, 28, 29, 30]. However, cases in which failure leads
 75 to unrecoverable states or yields unacceptable behavior may prohibit their use in deployment settings.
 76 Instead, we aim to detect the failing behavior of generative policies at test time, noting that our methods
 77 may be applied to the benefit of continual policy learning and online adaptation in the future.
 78 The existing literature on **out-of-distribution detectors** and **runtime monitoring** for learned models
 79 is highly diverse, spanning multiple categories of methods. Model-based methods (e.g., [31, 32])
 80 are not directly applicable to the model-free policies we consider. Some methods only pursue
 81 failure modes that are known *a priori* [33, 34, 35]. Many OOD detection works detect dissimilarity
 82 from training data via e.g., reconstruction [36, 37] or embedding similarity [6, 7], however, visual
 83 differences may not always result in policy failure. Other methods directly quantify epistemic
 84 uncertainty [38, 39, 40], but come with considerable computational expense or cannot be applied to
 85 diffusion models due to the iterative denoising process. Most related to our approach are algorithms
 86 that use consistency checks across sensing modalities and time [41, 42]. Different from these, we
 87 directly monitor the consistency of a learned policy’s action distributions to detect closed-failure.
 88 There is growing interest in the use of **Foundation Models** [43] toward increasing robustness in robotic
 89 systems. Large Language Models are used to detect anomalies [44, 45] and to replan under execution

failures [46, 47, 48, 45]. Reward models in the form of visual representations [49, 50] or VLMs [51] could be repurposed for failure detection by thresholding predicted rewards. However, [51] shows additional fine-tuning is required to obtain reliable reward estimates. Most closely related to our use of VLMs is that of Du et al. [52]. We highlight three key differences *w.r.t.* this work. First, we focus on zero-shot assessment with VLMs, whereas they fine-tune VLMs on human annotated datasets on the order of 10^5 trajectories. Second, while we seek to detect policy failure amidst task execution, they classify success at the end of an episode. Third, we consider the system-level role of VLMs operating alongside policy-level failure detectors, and as such, assign each detector to a specified category of failure.

3 Problem Setup

Failure Detection The goal of this work is to detect when a generative robot policy fails to complete its task. Consider a policy $\pi(a|s)$ that operates within a finite-horizon Markov Decision Process (MDP): a 5-tuple $\langle \mathcal{S}, \mathcal{A}, T, R, H \rangle$, where \mathcal{S} and \mathcal{A} are the state and action spaces, $T(s'|s, a)$ is the transition model, $R(s, a, s')$ is the reward model, and H is the MDP horizon. Given an initial state s_0 representative of a new test scenario, executing the policy for t timesteps produces a trajectory $\tau_t = (s_0, a_0, \dots, s_t)$. The trajectory’s *return* is defined as the cumulative sum of rewards: $R(\tau_t) = \sum_{t'=0}^{t-1} R(s_{t'}, a_{t'}, s_{t'+1})$.

We define **policy failure** simply in terms of task completion. More formally, given a defined success threshold R_τ , the policy fails if the return on its trajectory τ_t does not exceed the success threshold within the MDP horizon: $R(\tau_t) < R_\tau$ where $t \geq H$. In the simplest case, the success threshold R_τ equals 1, and the reward model $R(s, a, s')$ equals 1 *iff* the task is complete at state s' . For example, if the robot is tasked with picking up a cup and receives a reward of 1 only once the cup is firmly grasped.

In this work, we define **failure detection** as the task of detecting in which trajectory τ_t the policy will fail at the earliest possible timestep t^1 . To do so, we aim to construct a failure detector $f_\phi(\tau_t) \rightarrow \{\text{ok}, \text{failure}\}$ that, at each timestep t , can provide a classification as to whether the policy *will fail* if it continues executing for the remaining $H - t$ timesteps of the MPD. Here, ϕ denotes the parameters of the failure detector. For example, ϕ could represent a threshold on some quantity extracted from the trajectory (e.g., the maximum acceleration). Note that the failure detector makes its assessment solely based on the history of observed states and executed actions up to the current timestep t .

Furthermore, we assume a scenario in which a policy π is first trained, then validated on test cases where it is expected to perform reliably. This validation process yields a small dataset of M successful trajectories $\mathcal{D}_\tau = \{\tau^i\}_{i=1}^M$ that can be used to calibrate the parameters ϕ of the failure detector f_ϕ . Intuitively, the dataset \mathcal{D}_τ characterizes the nominal behavior of the policy within or near the distribution of states it has been trained on, which helps to ground the assessment of potentially OOD trajectories at test time.

We measure performance in terms of true positive rate (TPR), true negative rate (TNR), and detection time (DT). We count a true positive if the failure detector raises a warning at any timestep in a trajectory where the policy fails, the earliest of which counts as the detection time. We count a true negative if the failure detector never raises a warning in a trajectory where the policy succeeds.

Policy Formulation We consider the setting where the policy π is stochastic and predicts a sequence of actions for the next h timesteps. That is, the action sequence sampled at the t -th timestep, $a_t \sim \pi(\cdot|s_t)$, consists of h actions $a_t := (a_{t|t}, a_{t+1|t}, \dots, a_{t+h-1|t})$, where the notation $a_{t+i|t}$ denotes the action prediction for time $t+i$ generated at timestep t (as in [53]). The actions $a_{\cdot|t} \in \mathcal{A}$ may correspond to e.g., end-effector poses or velocities. To control the robot, we sample an action sequence and execute the first $k < h$ actions, $a_{t:t+k|t}$, after which we re-evaluate the policy at timestep $t+k$. We visualize this receding horizon rollout in Fig. 1. Notably, a_t and a_{t+k} contain actions that temporally overlap for $h - k$ timesteps (i.e., at $a_{t+k:t+h|t}$ and $a_{t+k:t+h|t+k}$).

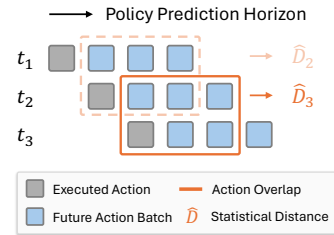


Figure 1: Action sequence prediction overlap during rollout.

¹Note that this is different from the detecting the timestep in which the policy “fails” by some criteria of failure. Defining the failure detection task in terms of task completion removes the need to specify explicit failure criteria for each task; the most general and intuitive failure criterion is whether or not the policy completes its task.

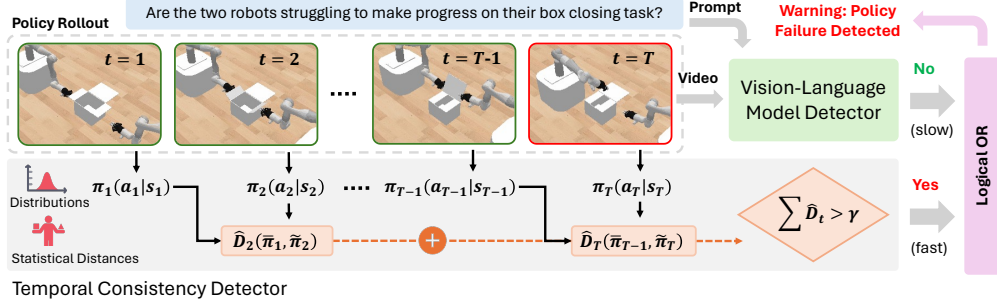


Figure 2: **Proposed failure detection framework.** The images depict a policy rollout on the Close Box domain for timesteps $t = 1, \dots, T$. Temporal Consistency Detector: At each timestep t , the state s_t is passed to the generative policy to obtain action distributions π_t between which statistical distances \hat{D}_t are computed to measure temporal consistency. The statistical distances are summed up to the current timestep T (as in Eq. 2) and thresholded by γ to detect policy failure. Vision-Language-Model (VLM) Detector: The VLM classifies whether the policy is failing to make progress on its task given a video up to timestep T and a description of the task. Execution stops if either detector raises a warning.

Several recently proposed policy architectures achieve state-of-the-art performance by sampling action sequences using generative models, to which our approach is generically applicable (§4). However, in this paper, we specify to the problem of failure detection for diffusion policies (DP) [1], which represent the policy distribution with a denoising diffusion probabilistic model (DDPM) [22]. In a DP, actions are generated by iteratively denoising an initially random action $a_t^N \sim \mathcal{N}(0, 1)$ over N steps as a_t^N, \dots, a_t^0 , where a_t^i with a superscript i denotes the generated action sequence at the i -th denoising iteration. In an imitation learning setting, the DP’s noise prediction network ϵ_θ is trained to predict the noise added to actions drawn from a dataset of expert demonstrations $\mathcal{D}_{\text{train}}$ by minimizing

$$\mathcal{L}_{\text{ddpm}} := \mathbb{E}_{(s, a^0) \sim \mathcal{D}_{\text{train}}, \epsilon^i, i} [\|\epsilon^i - \epsilon_\theta(\sqrt{\bar{\alpha}_i} a^0 + \sqrt{1 - \bar{\alpha}_i} \epsilon^i, s, i)\|^2], \quad (1)$$

where the constants $\bar{\alpha}_i$ depend on the chosen noising schedule of the diffusion process.

4 Proposed Approach

The failure behavior of a generative policy by OOD conditions can be highly diverse, and we therefore argue that the desiderata for a failure detector may vary between qualitative types of failures. In this work, we propose to split the failure detection task into two categories of failures.

The first is the detection of failures resulting from erratic policy behavior, which may cause a robot to end up in states that are difficult or costly to reset from, knock over objects, or lead to safety hazards. Therefore, it is important to detect erratic behavior as quickly as possible (§4.1). The second category is the detection of failures in which the policy struggles to make progress on its task (hereafter referred to as *task progression failures*) but does so in a temporally consistent manner. For example, the policy may confidently place an object in the wrong location. Here, we must observe the robot over a longer period of time to identify that the policy is not making progress towards task completion (§4.2).

The **key insight** of our approach is that it is trivial to combine failure detectors specified to these two complementary categories, thereby yielding an accurate overall failure detection pipeline while satisfying the detection requirements of each failure category. The full pipeline is visualized in Fig. 2.

4.1 STAC: Detecting Erratic Failures with Temporal Consistency

When a policy operates in nominal, in-distribution settings, it should complete its task in a temporally consistent manner. For example, a policy may plan to avoid an obstacle on the right or on the left, but not jitter between the two options. Moreover, as noted in [1], training a diffusion policy that predicts action sequences rather than individual actions encourages temporal consistency between action predictions.

Therefore, we propose to construct a quantifiable measure of temporal action consistency to detect whether the policy is behaving erratically, and hence, is likely to fail at the task. However, the

multi-modal distributional nature of DPs makes it difficult to directly compare two sampled actions $a_t \sim \pi(a_t|s_t)$ and $a_{t+k} \sim \pi(a_{t+k}|s_{t+k})$, e.g., throughout execution. This is because the actions may differ substantially along their prediction horizon when the policy commits or switches between different action modes, or simply due to randomness in sampling. Instead, we quantify erratic policy behavior with statistical measures of temporal action consistency (STAC, which we term our approach).

Let $\bar{\pi}_t := \pi(a_{t+k:t+h}|s_t)$ and $\tilde{\pi}_{t+k} := \pi(a_{t+k:t+h}|s_{t+k})$ be the marginal action distributions of the temporally overlapping actions between timesteps t and $t+k$. We compute the temporal consistency between two contiguous timesteps t and $t+k$ as $D(\bar{\pi}_t, \tilde{\pi}_{t+k}) \geq 0$, where D denotes the chosen statistical distance function (e.g., KL-divergence, maximum mean discrepancy). Due to the iterative denoising procedure of the DP, analytically computing a distance D is challenging, as evaluating the densities of $\bar{\pi}$ and $\tilde{\pi}$ requires marginalizing out the intermediate diffusion steps as well as the non-overlapping actions. Instead, we approximate D with its empirical counterpart \hat{D} by sampling a batch of action sequences (parallelized on a GPU) at each timestep t and $t+k$ rather than a single action sequence.

In addition to the use of a statistical distance between action distributions, we propose to take the cumulative sum of statistical distances along a trajectory as a measure of the overall temporal consistency in a policy rollout. At each time $t = jk$ with $j \in \{0, 1, \dots\}$, we compute the temporal consistency score as

$$\eta_t := \sum_{i=0}^{j-1} \hat{D}(\bar{\pi}_{ik}, \tilde{\pi}_{(i+1)k}). \quad (2)$$

Computing the consistency score in a cumulative manner has two advantages over thresholding the distance at each timestep individually. Firstly, it allows us to detect instances where the temporal consistency is marginally larger than usual throughout the episode (e.g., jitter). Secondly, it allows us to detect instances where the policy is temporally inconsistent more often than in nominal scenarios.

At runtime, we raise a failure warning at the moment that η_t exceeds a failure detection threshold γ , which we calibrate offline using the validation dataset of successful trajectories \mathcal{D}_τ . Here, we first compute the cumulative temporal consistency scores throughout the entirety of the lengths H_i of trajectories in \mathcal{D}_τ , yielding $\{\eta_{H_i}^i\}_{i=1}^M$. Then, we set the threshold γ to the $1 - \delta$ quantile of $\{\eta_{H_i}^i\}_{i=1}^M$, where $\delta \in (0, 1)$ is a hyperparameter. This ensures that the false positive rate (FPR), the probability that we raise a false alarm and terminate any trajectory that is i.i.d. with respect to \mathcal{D}_τ , is at most δ . We note that STAC can be extended to provide a formal guarantees on FPRs using recent results from conformal prediction theory [54, 55], and as such, we derive a comformal guarantee for STAC in §D.

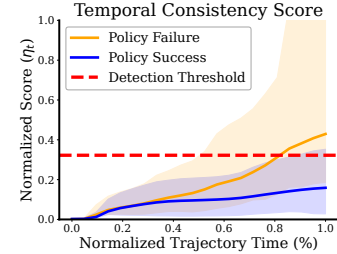


Figure 3: Temporal consistency scores grow faster when the policy fails. Error bars indicate the 5-th and 95-th score quantiles.

4.2 Detecting Task Progression Failures with VLMs

A policy operating out-of-distribution may not always fail by exhibiting erratic behavior that we can detect with STAC (§4.1). For example, suppose the policy confidently commits to the wrong plan or produces approximately constant outputs. In that case, the robot will fail to complete the task despite being temporally consistent in its actions. Detecting such failures requires an understanding as to whether or not the policy is progressing on its task, which necessitates a more comprehensive analysis of the robot’s behavior within the context of its task specification. Therefore, we propose to use VLMs to monitor the task progress of the policy by providing all of the robot’s image observations up to the current timestep as a video. We do so because recent work has shown that high-capacity VLMs possess robotics relevant knowledge and contextual reasoning abilities [56, 57, 52, 58, 51].

We formulate the detection of task progression failures as a chain-of-thought (CoT) [59] visual question answering (VQA) task [60, 61], reflecting current best practices in prompting. To capture a notion of *task progress*, the VLM must reason both across time and in the context of the policy’s task. Thus, we construct a prompt that contains a brief description of the robot’s task and the VLMs role as the runtime monitor, that is, to interfere when the robot is clearly failing to complete the task. We query the VLM online using the text prompt and the history of observed images (i.e., a video) $I_{0:t} := (I_0, I_{\nu k}, I_{2\nu k}, \dots, I_t)$

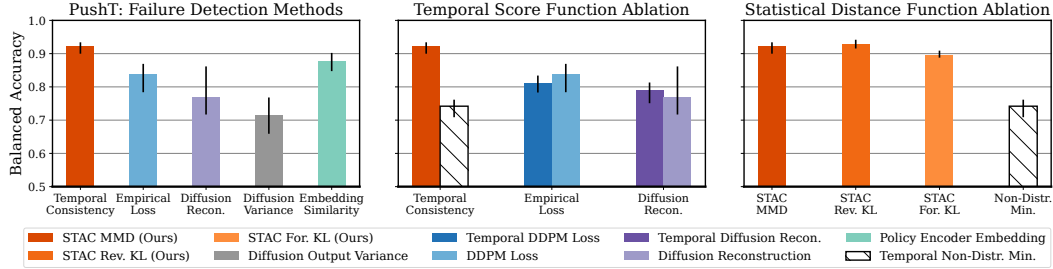


Figure 4: **Failure detection results on PushT** averaged over 3 random seeds (300 episodes total). **Left:** Our failure detector which measures the temporal consistency of a diffusion policy across time outperforms several families of out-of-distribution detectors. **Middle:** The best performance comes from measuring temporal consistency with statistical distance functions (denoted by **STAC**); simply augmenting baselines with temporal consistency does not meaningfully increase their performance. **Right:** Comparable performance is achieved across multiple choices of statistical distance functions.

up to the current timestep t . Here, the hyperparameter ν specifies the frequency of the images relative to the execution horizon k of the DP (§3). For additional details on the prompt, please see §A.2.

In brief, we elicit a three-step CoT response from the VLM that 1) describes the observed motion of the robot and task-relevant objects, 2) analyzes the video to identify if the robot is behaving incorrectly or whether it is making progress towards task completion within the episode time limit, and 3) concludes with a classification in $\{\text{ok}, \text{failure}\}$. At the time of writing, cloud-querying a state-of-the-art VLM with the robot’s observation video incurs significant latency (we used gpt-4o, with a mean response time of 14.0s). However, we emphasize that VLM inference latency is a lesser concern for detecting task progression failures because they are likely to occur at longer timescales and do not require urgent intervention. In contrast, we assign the rapid detection of erratic failures to STAC (§4.1).

5 Experiments

We conduct a series of experiments to test our failure detection framework. These experiments take place over multiple environments and host an extensive list of baselines that span five methodological categories for detecting distribution shift. The environments considered vary in terms of their data distribution (e.g., multi-modal, high-dimensional actions) and support different types of distribution shift (e.g., object scale, position), under which the behavior of the diffusion policies can be methodically studied. We refer to the Appendix for more a detailed description of our environments and baselines.

Environments. The **PushT** domain tasks a robot with pushing a planar “T”-shaped object into a goal configuration. This domain has been used to demonstrate the multi-modal properties of diffusion policies [1], and thus, we include it to evaluate the detection of failures under action multi-modality. The **Close Box** domain tasks two mobile manipulator robots with closing three lids of a box (see Fig. 2). Failures in this domain primarily result from erratic policy behavior when the robot e.g., collides with the box, fails to get under a lid, or drifts to OOD states. The **Cover Object** domain tasks two mobile robots to cover a rigid object with a deformable cloth. Failures in this domain result from the smooth but inadequate covering of the rigid object. Both **Close Box** and **Cover Box** present the challenge of a high-dimensional 14 degree-of-freedom action space. At test time, we generate OOD scenarios by randomizing a) the scale and dimensions of objects in **PushT** and **Close Box** and b) the position of the object in **Cover Object** beyond the randomizations contained in the policy’s demonstration data.

Baselines. We evaluate our approach (both STAC and the VLM runtime monitor) against baselines representative of multiple methodological categories in the OOD detection literature [4]. Intuitively, these categories represent different formulations of the failure detector’s score function, responsible for computing the per-timestep scores that are then summed to compute the trajectory score as in Eq. (2). We consider score functions based on the embedding similarity of observed states *w.r.t.* \mathcal{D}_τ [6], the reconstruction error of actions sampled from the DP [62], and the output variance of the DP. To strengthen the comparison, we introduce a new baseline that uses the DDPM loss (Eq. (1)) on a

Failure Detector	Close Box: In-Distribution			Close Box: Out-of-Distribution			Close Box: Combined			
	TPR \uparrow	TNR \uparrow	Det. Time (s) \downarrow	TPR \uparrow	TNR \uparrow	Det. Time (s) \downarrow	TPR \uparrow	TNR \uparrow	Accuracy \uparrow	
STAC	STAC MMD (Ours)	1.00	0.94	7.20	0.99	0.93	14.72	0.99	0.94	0.96
	STAC Rev. KL (Ours)	1.00	0.95	7.60	0.93	0.97	15.12	0.93	0.96	0.95
	STAC For. KL (Ours)	1.00	0.90	6.60	0.99	0.85	14.04	0.99	0.89	0.92
Embed.	Policy Encoder	0.25	0.98	16.27	1.00	0.00	1.59	0.94	0.70	0.78
	CLIP Pretrained	1.00	0.95	15.73	1.00	0.00	8.20	1.00	0.68	0.79
	ResNet Pretrained	1.00	0.95	17.87	1.00	0.00	15.51	1.00	0.68	0.79
Diffusion	Temporal Min.	1.00	0.97	5.00	1.00	0.27	12.35	1.00	0.77	0.85
	Diffusion Recon. [62]	0.33	0.95	13.60	0.40	1.00	17.08	0.37	0.96	0.76
	Temporal Diffusion Recon.	1.00	0.96	8.47	0.92	1.00	15.75	0.92	0.97	0.95
	DDPM Loss (Eq. (1))	1.00	0.90	8.27	1.00	0.94	14.54	1.00	0.91	0.94
	Temporal DDPM Loss	1.00	0.95	7.53	1.00	0.37	13.66	1.00	0.79	0.86
	Diffusion Output Variance	0.33	0.94	14.00	0.28	1.00	17.27	0.26	0.96	0.72
VLM	GPT-4o Image	1.00	0.00	24.00	1.00	0.00	24.00	1.00	0.00	0.32
	GPT-4o Video (Ours)	0.60	1.00	22.40	0.52	0.75	21.42	0.53	0.94	0.81

Table 1: **Detecting erratic policy failures in the Close Box domain.** Results are averaged over 3 random seeds. Our temporal consistency detector, STAC, accounts for when a policy fails (**high true positive rate**) and when it generalizes to out-of-distribution test cases (**high true negative rate**), in contrast to embedding-based methods that directly associate state atypicality with policy failure (**low true negative rate**). Select baselines that accurately detect erratic policy failure in this domain experience a decrease in performance under multi-modal conditions (i.e., **PushT**, as shown in Fig. 4), whereas STAC continues to exhibit **strong performance** across multiple domains. VLMs **must reason** over video to attain high true negative rates, as is necessary to combine them with STAC (see Fig. 5).

re-noised action sampled from the DP as the failure detector’s score function. Where applicable, we implement temporal consistency variants of baselines to ablate this design decision of our approach.

Evaluation Protocol. We train a DP for each environment and use standard settings for the DP’s prediction and execution horizon [1]. To increase the salience of distribution shift *w.r.t.* the position and scale of objects, we use point clouds as inputs the policy instead of RGB images (i.e., a 5% increase in object scale may not be salient in an image). We use the same calibration and evaluation protocol across all failure detection methods. That is, we calibrate detection thresholds to the 95-th quantile of scores in a dataset of 50 in-distribution rollouts $\mathcal{D}_\tau = \{\tau^i\}_{i=1}^{50}$ for each task. Finally, we report standard detection metrics including TPR, TNR, Mean Detection Time, Accuracy, and Balanced Accuracy (defined in §3). Further details on the DP architecture, training, and evaluation are provided in §B.3.

6 Results

We present the key findings of our experiments, with a particular focus on the design decisions of the proposed temporal consistency detector and the complementary nature of VLMs for failure detection.

Temporal consistency detects diffusion policy failures in multi-modal domains. Fig. 4 (Left) compares our temporal consistency detector, STAC, against the best performing method of each baseline category in the **PushT** domain. Here, STAC is the only method to achieve a balanced accuracy of over 90%, indicating that temporal consistency (or lack thereof) is strongly correlated with success (or failure). Alternative output metrics, such as the DP’s output variance (Diffusion Variance), do not perform well because both successes and failures can exhibit high variance outputs in multi-modal domains. Interestingly, the embedding similarity approach performs strongly in this domain, which indicates that state dissimilarity *w.r.t.* the calibration dataset happens to be correlated with failure. However, as we later show, this approach fails when the policy generalizes to out-of-distribution states.

Statistical measures of action similarity enables temporal consistency detection. Fig. 4 (Middle) ablates the design decisions of STAC. First, we observe that augmenting baselines with temporal consistency will at most marginally increase their performance. Second, using a non-statistical distance function (e.g., min. distance) to measure temporal action consistency performs worse than the baselines because it omits action multi-modality. Instead, we find that it is the combination of statistical distance functions with temporal consistency that yields the best result. This is corroborated in Fig. 4 (Right), where STAC performs comparably across common statistical distance functions like maximum mean discrepancy (MMD) with RBF kernels and KL-divergence via kernel density estimation.

STAC accounts for out-of-distribution failure and generalization. Results on the **Close Box** domains are shown in Table 1. STAC attains the highest accuracy in aggregate. However, two of our newly proposed baselines—using the DDPM loss (Eq. (1)) and a temporal reconstruction variant of [62]—also perform well, perhaps due to a decrease in action multi-modality relative to **PushT** (where performance gaps are more notable). Importantly, we observe that embedding similarity methods conflate OOD states with policy failure, resulting in false positives when the policy succeeds OOD. By monitoring the policy’s behavior, STAC effectively differentiates between OOD success and failure.

VLMs must reason across time. In Table 1, we find that a state-of-the-art VLM (gpt-4o) struggles to identify task success when given only a single image. Instead, it must observe the robot over the extent of a policy rollout to more accurately reason about task progression and changes in state (resulting in a 94% overall TNR). We note that while the erratic failures are time sensitive (e.g., the robot can damage the box if it collides with it), they are visually more subtle and therefore more complex for the VLM to interpret, whereas the robot takes more obviously wrong actions (e.g., stalling, clearly misplacing the cover) in the task progression failures (see Fig. 5). As expected, we observe that the VLM has a significantly slower detection times relative to STAC, further highlighting STAC’s value at quickly detecting erratic behavior. We refer to §C.2 for further discussion of Table 1’s VLM results.

VLMs complement STAC for system-level performance increases. We evaluate our full failure detection approach on distribution shifts that primarily lead to task progression failures for both the **Cover Object** and **Close Box** domains. The result is shown in Fig. 5. As expected, STAC achieves a low TPR (50%) when the policy fails in a temporally consistent manner, whereas the VLM (video) accurately detects task progression failures. As a result, combining STAC with the VLM achieves a >80% TPR whilst incurring only a 5% increase in FPR. The rise in detection time indicates that both STAC (fast) and the VLM (slow) are contributing to the detection of failures.

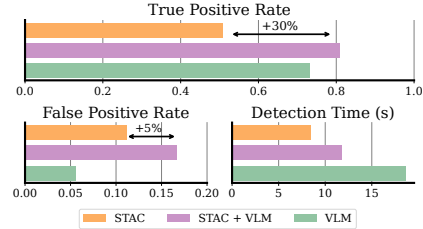


Figure 5: VLMs complement STAC to increase accuracy under distribution shifts causing task progression failures on **Close Box** and **Cover Object**.

Discussion. Holistic analysis of Table 1 and Fig. 5 highlights that we can easily combine STAC and the VLM to yield a performant detector for both erratic and task progression failures, particularly because both detectors achieve a high overall TNR. Importantly, the VLM outperforms the majority of baselines at accurately detecting task progression failures (results in §C.3), implying that the combination of STAC and the VLM is the best detector overall: Because all the baselines may 1) show low accuracy on one or both the erratic failure domains (i.e., the multi-modal **PushT** task or the **Close Box** task) or 2) yield a low TNR, it is unclear how to combine them with other detectors in a way that outperforms STAC+VLM.

7 Conclusion

In this work, we investigate the problem of failure detection for generative robot policies. We split the failure detection task into two complementary categories: 1) Erratic failures, which we detect by measuring the statistical change of a policy’s action distributions over time; 2) Task progression failures, where we use Vision-Language Models to assess whether the policy is consistently taking actions that do not solve the task. Our experiments highlight the importance of targeting complementary failure categories with specialized detectors. Future work includes the use of our framework to accelerate human-in-the-loop policy learning and facilitate online adaptation by summarizing failures.

Limitations. While categorizing erratic and task progression failures leads to accurate detection of the diverse failures modes of generative policies, the VLM still limits our framework’s efficiency. In the future, introducing additional categories or further partitioning existing ones might allow for more efficient failure detection and inform mitigation strategies. Furthermore, our approach does not provide formal guarantees of failure detection. However, providing such guarantees would require data of both successful and unsuccessful policy rollouts to calibrate the detector [55]. Finally, our approach is not targeted at predicting failures before they occur but instead focuses on detecting failures as they occur.

References

- [1] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. C. Burchfiel, and S. Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi:10.15607/RSS.2023.XIX.026.
- [2] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR, 2022.
- [3] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.
- [4] R. Sinha, A. Sharma, S. Banerjee, T. Lew, R. Luo, S. M. Richards, Y. Sun, E. Schmerling, and M. Pavone. A system-level view on out-of-distribution data in robotics. *arXiv preprint arXiv:2212.14020*, 2022. Available at <https://arxiv.org/abs/2212.14020>.
- [5] J. Yang, K. Zhou, Y. Li, and Z. Liu. Generalized out-of-distribution detection: A survey. *CoRR*, abs/2110.11334, 2021. URL <https://arxiv.org/abs/2110.11334>.
- [6] K. Lee, K. Lee, H. Lee, and J. Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/abdeb6f575ac5c6676b747bca8d09cc2-Paper.pdf.
- [7] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft. Deep one-class classification. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4393–4402. PMLR, 10–15 Jul 2018. URL <https://proceedings.mlr.press/v80/ruff18a.html>.
- [8] A. Sharma, N. Azizan, and M. Pavone. Sketching curvature for efficient out-of-distribution detection for deep neural networks. In *Uncertainty in artificial intelligence*, pages 1958–1967. PMLR, 2021.
- [9] J. Yang, C. Deng, J. Wu, R. Antonova, L. Guibas, and J. Bohg. Equivact: Sim(3)-equivariant visuomotor policies beyond rigid object manipulation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024.
- [10] H. Bharadhwaj, J. Vakil, M. Sharma, A. Gupta, S. Tulsiani, and V. Kumar. Roboagent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking, 2023.
- [11] A. Goyal, J. Xu, Y. Guo, V. Blukis, Y.-W. Chao, and D. Fox. Rvt: Robotic view transformer for 3d object manipulation. In *Conference on Robot Learning*, pages 694–710. PMLR, 2023.
- [12] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi:10.15607/RSS.2023.XIX.016.
- [13] C. Chi, Z. Xu, C. Pan, E. Cousineau, B. Burchfiel, S. Feng, R. Tedrake, and S. Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.
- [14] Z. Fu, T. Z. Zhao, and C. Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. In *arXiv*, 2024.

- [15] H. Walke, K. Black, A. Lee, M. J. Kim, M. Du, C. Zheng, T. Zhao, P. Hansen-Estruch, Q. Vuong, A. He, V. Myers, K. Fang, C. Finn, and S. Levine. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning (CoRL)*, 2023.
- [16] O. X.-E. Collaboration, A. O’Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, A. Tung, A. Bewley, A. Herzog, A. Irpan, A. Khazatsky, A. Rai, A. Gupta, A. Wang, A. Kolobov, A. Singh, A. Garg, A. Kembhavi, A. Xie, A. Brohan, A. Raffin, A. Sharma, A. Yavary, A. Jain, A. Balakrishna, A. Wahid, B. Burgess-Limerick, B. Kim, B. Schölkopf, B. Wulfe, B. Ichter, C. Lu, C. Xu, C. Le, C. Finn, C. Wang, C. Xu, C. Chi, C. Huang, C. Chan, C. Agia, C. Pan, C. Fu, C. Devin, D. Xu, D. Morton, D. Driess, D. Chen, D. Pathak, D. Shah, D. Büchler, D. Jayaraman, D. Kalashnikov, D. Sadigh, E. Johns, E. Foster, F. Liu, F. Ceola, F. Xia, F. Zhao, F. V. Frueger, F. Stulp, G. Zhou, G. S. Sukhatme, G. Salhotra, G. Yan, G. Feng, G. Schiavi, G. Berseth, G. Kahn, G. Yang, G. Wang, H. Su, H.-S. Fang, H. Shi, H. Bao, H. B. Amor, H. I. Christensen, H. Furuta, H. Walke, H. Fang, H. Ha, I. Mordatch, I. Radosavovic, I. Leal, J. Liang, J. Abou-Chakra, J. Kim, J. Drake, J. Peters, J. Schneider, J. Hsu, J. Bohg, J. Bingham, J. Wu, J. Gao, J. Hu, J. Wu, J. Wu, J. Sun, J. Luo, J. Gu, J. Tan, J. Oh, J. Wu, J. Lu, J. Yang, J. Malik, J. Silvério, J. Hejna, J. Booher, J. Tompson, J. Yang, J. Salvador, J. J. Lim, J. Han, K. Wang, K. Rao, K. Pertsch, K. Hausman, K. Go, K. Gopalakrishnan, K. Goldberg, K. Byrne, K. Oslund, K. Kawaharazuka, K. Black, K. Lin, K. Zhang, K. Ehsani, K. Lekkala, K. Ellis, K. Rana, K. Srinivasan, K. Fang, K. P. Singh, K.-H. Zeng, K. Hatch, K. Hsu, L. Itti, L. Y. Chen, L. Pinto, L. Fei-Fei, L. Tan, L. J. Fan, L. Ott, L. Lee, L. Weihs, M. Chen, M. Lepert, M. Memmel, M. Tomizuka, M. Itkina, M. G. Castro, M. Spero, M. Du, M. Ahn, M. C. Yip, M. Zhang, M. Ding, M. Heo, M. K. Srirama, M. Sharma, M. J. Kim, N. Kanazawa, N. Hansen, N. Heess, N. J. Joshi, N. Suenderhauf, N. Liu, N. D. Palo, N. M. M. Shafiullah, O. Mees, O. Kroemer, O. Bastani, P. R. Sanketi, P. T. Miller, P. Yin, P. Wohlhart, P. Xu, P. D. Fagan, P. Mitrano, P. Sermanet, P. Abbeel, P. Sundaresan, Q. Chen, Q. Vuong, R. Rafailov, R. Tian, R. Doshi, R. Mart’ in-Mart’ in, R. Baijal, R. Scalise, R. Hendrix, R. Lin, R. Qian, R. Zhang, R. Mendonca, R. Shah, R. Hoque, R. Julian, S. Bustamante, S. Kirmani, S. Levine, S. Lin, S. Moore, S. Bahl, S. Dass, S. Sonawani, S. Song, S. Xu, S. Haldar, S. Karamcheti, S. Adebola, S. Guist, S. Nasiriany, S. Schaal, S. Welker, S. Tian, S. Ramamoorthy, S. Dasari, S. Belkhale, S. Park, S. Nair, S. Mirchandani, T. Osa, T. Gupta, T. Harada, T. Matsushima, T. Xiao, T. Kollar, T. Yu, T. Ding, T. Davchev, T. Z. Zhao, T. Armstrong, T. Darrell, T. Chung, V. Jain, V. Vanhoucke, W. Zhan, W. Zhou, W. Burgard, X. Chen, X. Chen, X. Wang, X. Zhu, X. Geng, X. Liu, X. Liangwei, X. Li, Y. Pang, Y. Lu, Y. J. Ma, Y. Kim, Y. Chebotar, Y. Zhou, Y. Zhu, Y. Wu, Y. Xu, Y. Wang, Y. Bisk, Y. Dou, Y. Cho, Y. Lee, Y. Cui, Y. Cao, Y.-H. Wu, Y. Tang, Y. Zhu, Y. Zhang, Y. Jiang, Y. Li, Y. Li, Y. Iwasawa, Y. Matsuo, Z. Ma, Z. Xu, Z. J. Cui, Z. Zhang, Z. Fu, and Z. Lin. Open X-Embodiment: Robotic learning datasets and RT-X models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024.
- [17] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, P. D. Fagan, J. Hejna, M. Itkina, M. Lepert, Y. J. Ma, P. T. Miller, J. Wu, S. Belkhale, S. Dass, H. Ha, A. Jain, A. Lee, Y. Lee, M. Memmel, S. Park, I. Radosavovic, K. Wang, A. Zhan, K. Black, C. Chi, K. B. Hatch, S. Lin, J. Lu, J. Mercat, A. Rehman, P. R. Sanketi, A. Sharma, C. Simpson, Q. Vuong, H. R. Walke, B. Wulfe, T. Xiao, J. H. Yang, A. Yavary, T. Z. Zhao, C. Agia, R. Baijal, M. G. Castro, D. Chen, Q. Chen, T. Chung, J. Drake, E. P. Foster, J. Gao, D. A. Herrera, M. Heo, K. Hsu, J. Hu, D. Jackson, C. Le, Y. Li, K. Lin, R. Lin, Z. Ma, A. Maddukuri, S. Mirchandani, D. Morton, T. Nguyen, A. O’Neill, R. Scalise, D. Seale, V. Son, S. Tian, E. Tran, A. E. Wang, Y. Wu, A. Xie, J. Yang, P. Yin, Y. Zhang, O. Bastani, G. Berseth, J. Bohg, K. Goldberg, A. Gupta, A. Gupta, D. Jayaraman, J. J. Lim, J. Malik, R. Martín-Martín, S. Ramamoorthy, D. Sadigh, S. Song, J. Wu, M. C. Yip, Y. Zhu, T. Kollar, S. Levine, and C. Finn. Droid: A large-scale in-the-wild robot manipulation dataset. 2024.
- [18] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, T. Jackson, S. Jesmonth, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal, K.-H. Lee, S. Levine, Y. Lu, U. Malla, D. Manjunath,

- 426 I. Mordatch, O. Nachum, C. Parada, J. Peralta, E. Perez, K. Pertsch, J. Quiambao, K. Rao, M. S.
427 Ryoo, G. Salazar, P. R. Sanketi, K. Sayed, J. Singh, S. Sontakke, A. Stone, C. Tan, H. Tran,
428 V. Vanhoucke, S. Vega, Q. H. Vuong, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich. RT-1:
429 Robotics Transformer for Real-World Control at Scale. In *Proceedings of Robotics: Science and*
430 *Systems*, Daegu, Republic of Korea, July 2023. doi:10.15607/RSS.2023.XIX.025.
- 431 [19] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid,
432 Q. Vuong, V. Vanhoucke, H. Tran, R. Soricut, A. Singh, J. Singh, P. Sermanet, P. R. Sanketi,
433 G. Salazar, M. S. Ryoo, K. Reymann, K. Rao, K. Pertsch, I. Mordatch, H. Michalewski, Y. Lu,
434 S. Levine, L. Lee, T.-W. E. Lee, I. Leal, Y. Kuang, D. Kalashnikov, R. Julian, N. J. Joshi, A. Irpan,
435 B. Ichter, J. Hsu, A. Herzog, K. Hausman, K. Gopalakrishnan, C. Fu, P. Florence, C. Finn, K. A.
436 Dubey, D. Driess, T. Ding, K. M. Choromanski, X. Chen, Y. Chebotar, J. Carbajal, N. Brown,
437 A. Brohan, M. G. Arenas, and K. Han. Rt-2: Vision-language-action models transfer web
438 knowledge to robotic control. In J. Tan, M. Toussaint, and K. Darvish, editors, *Proceedings of The*
439 *7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*,
440 pages 2165–2183. PMLR, 06–09 Nov 2023. URL [https://proceedings.mlr.press/v229/](https://proceedings.mlr.press/v229/zitkovich23a.html)
441 [zitkovich23a.html](https://proceedings.mlr.press/v229/zitkovich23a.html).
- 442 [20] Octo Model Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, C. Xu,
443 J. Luo, T. Kreiman, Y. Tan, L. Y. Chen, P. Sanketi, Q. Vuong, T. Xiao, D. Sadigh, C. Finn, and
444 S. Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science*
445 *and Systems*, Delft, Netherlands, 2024.
- 446 [21] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang. A tutorial on energy-based learning.
447 *Predicting structured data*, 1(0), 2006.
- 448 [22] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural*
449 *information processing systems*, 33:6840–6851, 2020.
- 450 [23] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative
451 modeling through stochastic differential equations. In *International Conference on Learning*
452 *Representations*, 2021. URL <https://openreview.net/forum?id=PxTIG12RRHS>.
- 453 [24] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the*
454 *thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR
455 Workshop and Conference Proceedings, 2010.
- 456 [25] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to
457 no-regret online learning. In *Proceedings of the fourteenth international conference on artificial*
458 *intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.
- 459 [26] A. Bajcsy, D. P. Losey, M. K. O’Malley, and A. D. Dragan. Learning from physical human
460 corrections, one feature at a time. In *Proceedings of the 2018 ACM/IEEE International Conference*
461 *on Human-Robot Interaction*, pages 141–149, 2018.
- 462 [27] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer. Hg-dagger: Interactive
463 imitation learning with human experts. In *2019 International Conference on Robotics and*
464 *Automation (ICRA)*, pages 8077–8083. IEEE, 2019.
- 465 [28] A. Mandlekar, D. Xu, R. Martín-Martín, Y. Zhu, L. Fei-Fei, and S. Savarese. Human-in-
466 the-loop imitation learning using remote teleoperation. *CoRR*, abs/2012.06733, 2020. URL
467 <https://arxiv.org/abs/2012.06733>.
- 468 [29] R. Hoque, A. Balakrishna, E. Novoseller, A. Wilcox, D. S. Brown, and K. Goldberg. Thriftydagger:
469 Budget-aware novelty and risk gating for interactive imitation learning. In A. Faust, D. Hsu,
470 and G. Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164
471 of *Proceedings of Machine Learning Research*, pages 598–608. PMLR, 08–11 Nov 2022. URL
472 <https://proceedings.mlr.press/v164/hoque22a.html>.

- [30] Y. Cui, S. Karamcheti, R. Palleti, N. Shivakumar, P. Liang, and D. Sadigh. No, to the right: Online language corrections for robotic manipulation via shared autonomy. In *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, pages 93–101, 2023.
- [31] A. Marco, E. Morley, and C. J. Tomlin. Out of distribution detection via domain-informed gaussian process state space models. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pages 5487–5493, 2023. doi:10.1109/CDC49753.2023.10383566.
- [32] H. Liu, S. Dass, R. Martín-Martín, and Y. Zhu. Model-based runtime monitoring with interactive imitation learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.
- [33] A. Farid, D. Snyder, A. Ren, and A. Majumdar. Failure Prediction with Statistical Guarantees for Vision-Based Robot Control. In *Proceedings of Robotics: Science and Systems*, New York City, NY, USA, June 2022. doi:10.15607/RSS.2022.XVIII.042.
- [34] S. Daftry, S. Zeng, J. A. Bagnell, and M. Hebert. Introspective perception: Learning to predict failures in vision systems. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1743–1750, 2016. URL <https://api.semanticscholar.org/CorpusID:15394788>.
- [35] S. Rabiee and J. Biswas. Ivoa: Introspective vision for obstacle avoidance. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1230–1235, 2019. URL <https://api.semanticscholar.org/CorpusID:67855498>.
- [36] C. Richter and N. Roy. Safe visual navigation via deep learning and novelty detection. 2017.
- [37] L. Ruff, J. R. Kauffmann, R. A. Vandermeulen, G. Montavon, W. Samek, M. Kloft, T. G. Dietterich, and K.-R. Müller. A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*, 109(5):756–795, 2021. doi:10.1109/JPROC.2021.3052449.
- [38] J. Z. Liu, Z. Lin, S. Padhy, D. Tran, T. Bedrax-Weiss, and B. Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness, 2020.
- [39] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR. URL <https://proceedings.mlr.press/v48/gal16.html>.
- [40] A. Sharma, N. Azizan, and M. Pavone. Sketching curvature for efficient out-of-distribution detection for deep neural networks. In C. de Campos and M. H. Maathuis, editors, *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pages 1958–1967. PMLR, 27–30 Jul 2021. URL <https://proceedings.mlr.press/v161/sharma21a.html>.
- [41] M. A. Lee, M. Tan, Y. Zhu, and J. Bohg. Detect, reject, correct: Crossmodal compensation of corrupted sensors. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 909–916. IEEE, 2021.
- [42] P. Antonante, D. I. Spivak, and L. Carlone. Monitoring and diagnosability of perception systems. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 168–175, 2021. doi:10.1109/IROS51168.2021.9636497.
- [43] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.
- [44] A. Elhafsi, R. Sinha, C. Agia, E. Schmerling, I. A. Nesnas, and M. Pavone. Semantic anomaly detection with large language models. *Autonomous Robots*, 47(8):1035–1055, 2023.

- [45] R. Sinha, A. Elhafsi, C. Agia, M. Foutter, E. Schmerling, and M. Pavone. Real-time anomaly detection and reactive planning with large language models. In *Robotics: Science and Systems*, 2024.
- [46] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Thompson, I. Mordatch, Y. Chebotar, P. Sermanet, T. Jackson, N. Brown, L. Luu, S. Levine, K. Hausman, and b. ichter. Inner monologue: Embodied reasoning through planning with language models. In K. Liu, D. Kulic, and J. Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 1769–1782. PMLR, 14–18 Dec 2023. URL <https://proceedings.mlr.press/v205/huang23c.html>.
- [47] Z. Liu, A. Bahety, and S. Song. Reflect: Summarizing robot experiences for failure explanation and correction. In J. Tan, M. Toussaint, and K. Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 3468–3484. PMLR, 06–09 Nov 2023. URL <https://proceedings.mlr.press/v229/liu23g.html>.
- [48] M. Skreta, Z. Zhou, J. L. Yuan, K. Darvish, A. Aspuru-Guzik, and A. Garg. Replan: Robotic replanning with perception and language models. *arXiv preprint arXiv:2401.04157*, 2024.
- [49] Y. J. Ma, S. Sodhani, D. Jayaraman, O. Bastani, V. Kumar, and A. Zhang. VIP: Towards universal visual reward and representation via value-implicit pre-training. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=YJ7o2wetJ2>.
- [50] Y. Cui, S. Niekum, A. Gupta, V. Kumar, and A. Rajeswaran. Can foundation models perform zero-shot task specification for robot manipulation? In *Learning for dynamics and control conference*, pages 893–905. PMLR, 2022.
- [51] J. Yang, M. S. Mark, B. Vu, A. Sharma, J. Bohg, and C. Finn. Robot fine-tuning made easy: Pre-training rewards and policies for autonomous real-world reinforcement learning, 2023.
- [52] Y. Du, K. Konyushkova, M. Denil, A. Raju, J. Landon, F. Hill, N. de Freitas, and S. Cabi. Vision-language models as success detectors. In S. Chandar, R. Pascanu, H. Sedghi, and D. Precup, editors, *Proceedings of The 2nd Conference on Lifelong Learning Agents*, volume 232 of *Proceedings of Machine Learning Research*, pages 120–136. PMLR, 22–25 Aug 2023. URL <https://proceedings.mlr.press/v232/du23b.html>.
- [53] F. Borrelli, A. Bemporad, and M. Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.
- [54] A. N. Angelopoulos and S. Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511*, 2021.
- [55] R. Luo, S. Zhao, J. Kuck, B. Ivanovic, S. Savarese, E. Schmerling, and M. Pavone. Sample-efficient safety assurances using conformal prediction. In S. M. LaValle, J. M. O’Kane, M. Otte, D. Sadigh, and P. Tokekar, editors, *Algorithmic Foundations of Robotics XV*, pages 149–169, Cham, 2023. Springer International Publishing. ISBN 978-3-031-21090-7.
- [56] S. Nasiriany, F. Xia, W. Yu, T. Xiao, J. Liang, I. Dasgupta, A. Xie, D. Driess, A. Wahid, Z. Xu, et al. Pivot: Iterative visual prompting elicits actionable knowledge for vlms. *arXiv preprint arXiv:2402.07872*, 2024.
- [57] J. Gao, B. Sarkar, F. Xia, T. Xiao, J. Wu, B. Ichter, A. Majumdar, and D. Sadigh. Physically grounded vision-language models for robotic manipulation. 2024.
- [58] X. Li, M. Liu, H. Zhang, C. Yu, J. Xu, H. Wu, C. Cheang, Y. Jing, W. Zhang, H. Liu, H. Li, and T. Kong. Vision-language foundation models as effective robot imitators. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=lFYj0oibGR>.

- [59] J. Wei, X. Wang, D. Schuurmans, M. Bosma, brian ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou. Chain of thought prompting elicits reasoning in large language models. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQ1MeSB_J.
- [60] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015.
- [61] J. Xu, T. Mei, T. Yao, and Y. Rui. Msr-vtt: A large video description dataset for bridging video and language. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5288–5296, 2016.
- [62] M. S. Graham, W. H. Pinaya, P.-D. Tudosiu, P. Nachev, S. Ourselin, and J. Cardoso. Denoising diffusion models for out-of-distribution detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2947–2956, 2023.
- [63] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.
- [64] A. Gretton, R. Herbrich, A. Smola, O. Bousquet, B. Schölkopf, et al. Kernel methods for measuring independence. 2005.
- [65] K. Muandet, K. Fukumizu, B. Sriperumbudur, B. Schölkopf, et al. Kernel mean embedding of distributions: A review and beyond. *Foundations and Trends® in Machine Learning*, 10(1-2): 1–141, 2017.
- [66] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.
- [67] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [68] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.
- [69] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.
- [70] R. Luo, R. Sinha, Y. Sun, A. Hindy, S. Zhao, S. Savarese, E. Schmerling, and M. Pavone. Online distribution shift detection via recency prediction. In *Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA)*, Yokohama, Japan, 2024. IEEE.

597 **Appendix Overview: Unpacking Failure Modes of Generative Policies**

598 The appendix offers additional details with respect to the implementation of our failure detection
599 framework (§A), the experiments conducted (§B), along with extended results and analysis (§C),
600 and finally, supporting derivations (§D) for our proposed detectors. **Qualitative results** and a **video**
601 **abstract** are made available at sites.google.com/view/detecting-policy-failure.

602	Appendix A Method Details	16
603	A.1 Temporal Consistency Detection with STAC	16
604	A.2 Runtime Monitoring with Vision-Language Models	17
605	Appendix B Experiment Details	20
606	B.1 Environments	20
607	B.2 Baselines	20
608	B.3 Evaluation Protocol	25
609	Appendix C Additional Results	27
610	C.1 PushT Ablation	27
611	C.2 Vision-Language Model Ablation	27
612	C.3 Extended Discussion	28
613	Appendix D Derivations	30

614 A Method Details

615 As shown in Fig. 2, our proposed failure detection framework consists of the parallel operation of
 616 two complementary failure detectors, each assigned to the detection of a particular failure category of
 617 generative policies. The first is a temporal consistency detector that monitors for erratic policy behavior
 618 via statistical temporal action consistency (STAC) measures. The second is a Vision-Language Model
 619 (VLM) that monitors for failure of the policy to make progress on its task. In this section, we provide
 620 additional details *w.r.t.* the implementation of STAC (§A.1) and the VLM runtime monitor (§A.2).

621 A.1 Temporal Consistency Detection with STAC

622 **Background** To summarize §3, STAC assumes the use of a stochastic policy π that, at each policy
 623 inference timestep t , predicts an action sequence for the next h timesteps as $a_{t:t+h}|t \sim \pi(\cdot|s_t)$, executes
 624 the first k actions $a_{t:t+k}|t$, before re-evaluating the policy at timestep $t+k$. Here, range subscripts
 625 denote a sequence of actions. Between two contiguous inference timesteps t and $t+k$, sampled action
 626 sequences $a_{t+k:t+h}|t$ and $a_{t+k:t+h}|t+k$ (both in $\mathbb{R}^{(h-k) \times |\mathcal{A}|}$) overlap for $h-k$ timesteps. At a high-
 627 level, STAC seeks to quantify *how much* a generative policy’s action distributions are changing over
 628 time. It does this by computing statistical distances between the distributions of overlapping actions,
 629 i.e., given $\bar{\pi}_t := \pi(a_{t+k:t+h}|t|s_t)$ and $\tilde{\pi}_{t+k} := \pi(a_{t+k:t+h}|t+k|s_{t+k})$, we compute $D(\bar{\pi}_t, \tilde{\pi}_{t+k})$.

630 **Hypothesis** Our central hypothesis is that large statistical distances correlate with downstream policy
 631 failure. Intuitively, a predictive policy can be likened to possessing an internal world model that
 632 simulates how robot actions affect environment states. When the policy is in-distribution, we expect
 633 this world model to be accurate, thus resulting in smaller statistical distances. More concretely, if
 634 the policy’s internal model of state s_{t+k} at timestep t coincides with the actual observed state s_{t+k} at
 635 timestep $t+k$, the distribution of actions $\tilde{\pi}_{t+k}$ should be well-represented by the distribution $\bar{\pi}_t$. As a
 636 result, the distance $D(\bar{\pi}_t, \tilde{\pi}_{t+k})$ will be small (for the right choice of statistical distance function D).
 637 Conversely, when the policy is out-of-distribution (OOD), its internal model of state s_{t+k} at timestep t
 638 may be inaccurate, yielding a divergence between $\bar{\pi}_t$ and $\tilde{\pi}_{t+k}$ and a larger statistical distance.

639 **Implementation Details** As mentioned in §4.1, we propose to approximate $D(\bar{\pi}_t, \tilde{\pi}_{t+k})$ with an
 640 empirical distance function \hat{D} instead of computing it analytically, as doing so presents the challenge of
 641 marginalizing out both the non-overlapping actions (between timesteps t and $t+k$) and the intermediate
 642 steps of the diffusion process [63]. We found the following approximations to work well in practice:

- 643 • Maximum Mean Discrepancy (MMD) with radial basis function (RBF) kernels. We compute

$$\begin{aligned} \hat{D}(\bar{\pi}_t, \tilde{\pi}_{t+k}) = & \mathbb{E}_{a_t, a'_t \sim \bar{\pi}_t} [k(a_t, a'_t)] + \mathbb{E}_{a_{t+k}, a'_{t+k} \sim \tilde{\pi}_{t+k}} [k(a_{t+k}, a'_{t+k})] \\ & - 2\mathbb{E}_{a_t \sim \bar{\pi}_t, a_{t+k} \sim \tilde{\pi}_{t+k}} [k(a_t, a_{t+k})], \quad \text{where} \quad k(x, y; \beta_1) = \exp\left(-\frac{\|x-y\|^2}{\beta_1}\right). \end{aligned}$$

644 That is, $k: \mathbb{R}^{(h-k) \times |\mathcal{A}|} \times \mathbb{R}^{(h-k) \times |\mathcal{A}|} \rightarrow \mathbb{R}$ computes the similarity between two overlapping
 645 action sequences, and β_1 denotes the bandwidth of the RBF kernel. The expectations are
 646 taken over a batch of B action sequences sampled from the generative policy.

- 647 • Forward KL-divergence via Kernel Density Estimation (KDE) of the policy distributions:

$$\hat{D}(\bar{\pi}_t, \tilde{\pi}_{t+k}) = \mathbb{E}_{a_{t+k} \sim \tilde{\pi}_{t+k}} \left[\log \frac{p(a_{t+k})}{q(a_{t+k})} \right],$$

648 where p and q are KDEs of $\tilde{\pi}_{t+k}$ and $\bar{\pi}_t$ fit on a batch of B action sequences sampled from each
 649 distribution, respectively. As before, we use Gaussian RBF kernels of the form $k(x, y; \beta_2)$,
 650 where β_2 denotes the bandwidth of the RBF kernels used for KDE.

- Reverse KL-divergence via KDE of the policy distributions:

$$\hat{D}(\tilde{\pi}_t, \tilde{\pi}_{t+k}) = \mathbb{E}_{a_t \sim \tilde{\pi}_t} \left[\log \frac{p(a_t)}{q(a_t)} \right],$$

where p and q are KDEs of $\tilde{\pi}_t$ and $\tilde{\pi}_{t+k}$ fit on a batch of B action sequences sampled from each distribution, respectively, and all other parameters follow the forward KL definitions.

The batch size B , MMD bandwidth β_1 , and KDE bandwidth β_2 are hyperparameters that we select for a given environment. As expected, we found that larger batch sizes are necessary for accurate mean embeddings and density estimates in domains with higher degrees of multi-modality (i.e., **PushT**). We also found that using either default settings or dynamic calibration techniques are sufficient to obtain suitable MMD and KDE bandwidth parameters β_1 and β_2 , respectively. For example, setting β_2 in proportion to the maximum eigenvalue of the covariance of overlapping actions $a_{t+k:t+h}$ sampled from $\tilde{\pi}_t$ and $\tilde{\pi}_{t+k}$. Further details on selecting hyperparameters are provided in [Table 2](#).

Hyperparameters	PushT Domain (\uparrow Multi-Modal)	Mobile Manip. Domains (\downarrow Multi-Modal)
MMD + KDE batch size (B)	256	32
MMD bandwidth (β_1)	Median Heuristic [64, 65]	$1.0/ \mathcal{A} $
KDE bandwidth (β_2)	$\sqrt{\lambda_{\max}(\text{Cov}(a_{t+k:t+h}))}$	1.0
Policy action space (\mathcal{A})	Linear Velocity	Linear + Angular Velocity
Policy prediction horizon (h)	16	16
Policy execution horizon (k)	8	4

Table 2: Hyperparameters settings for temporal consistency detection with STAC.

Additional Design Choices There are several additional settings that one could adjust to increase STAC’s detection performance on their task. First, filtering components of the policy’s action space that are either noisy or discrete can increase the quality of the statistical distance estimates. For example, the policy’s action space in our bi-manual mobile robotic manipulator domains (i.e., **Close Box** and **Cover Object**) include end-effector linear and angular velocities, as well as a binary gripper command. However, when computing statistical distances, we omit all binary gripper commands. Next, reducing the execution horizon k of the generative policy to compare action distributions that are closer in time can mitigate excessively large statistical distances in highly dynamic or stochastic environments. Likewise, comparing action distributions over a shorter prediction horizon h may be suitable if the tails of predicted action sequences e.g., exhibit high variance. [Table 2](#) summarizes our design choices.

A.2 Runtime Monitoring with Vision-Language Models

In this section, we provide details surrounding the implementation of our VLM runtime monitor, after which we provide the prompt templates used in our experiments.

Implementation Details We use OpenAI’s gpt-4o multi-modal model to reason about the behavior of the policy and detect failures. To do so, we provide the VLM with both a parsed text prompt describing the monitoring task and the video recorded by the robot’s camera system up to the current timestep. Specifically, as described in §4.2, we query the VLM online at each timestep aligned with the robot’s execution horizon (i.e., for each $t = jk$ for $j \in \{0, 1, \dots\}$) using the history of observed images $I_{0:t} := (I_0, I_{\nu k}, I_{2\nu k}, \dots, I_t)$ up to the current timestep t . Here, the hyperparameter ν specifies the frequency of the images relative to the execution horizon k of the DP (§3) for generality, as the video may be captured at a much higher frame rate than the diffusion policy’s execution rate. In our implementation, we simply set $\nu = 1$ and found that this provided sufficient granularity for the model to identify the robot’s motion. To format the video so that it can be passed to the model, we follow the [OpenAI API template](#) by converting each individual image frame in the video to a jpg file, encoding the jpg with a base64 encoding, and then converting them to utf-8 strings. We then prompt the model by providing an input tuple consisting of the text prompt and each of the individually encoded images.

687 We do so because this is the recommended method of providing gpt-4o with video; at the time of
688 writing, the OpenAI Python API does not support direct video (mp4) inputs.

689 The prompt template consists of three parts. First is a brief description of the model’s role as the runtime
690 monitor of a manipulator robot, which the VLM must execute by analyzing the attached video of the
691 robot’s current progress. Second is a description of the task that the robot has to complete, as well
692 as the total amount of time that has elapsed relative to the episode time limit (corresponding to the
693 MDP horizon in §3). We make sure that the task description is sufficiently detailed, so that there is no
694 ambiguity over what the expected behavior of the robot is and what constitutes task completion. For
695 example, we specify that the object must be *fully* covered by the blanket in the **Cover Object** domain,
696 and that the robot must close all three of the box’s lids in the **Close Box** domain. We also found that it
697 is necessary to specify the elapsed time, since we query the VLM at each timestep within an episode
698 rather than making a success/failure classification after the episode completes. Online monitoring
699 requires the VLM to differentiate whether the robot is still in-progress of completing the task correctly
700 (in which case the current video represents partial progress), or whether the robot will fail to complete
701 the task (by e.g., stalling in a partially completed state). Differentiating between partial progress and
702 task failure can be ambiguous for a slow moving robot, and thus, providing the model with the current
703 elapsed time serves as a reference to gauge whether or not the rate at which the robot is executing the
704 task will result in a timely task completion. The third component of the prompt contains instructions
705 to elicit a chain-of-thought response [59], ensuring that the VLM describes and analyzes the robot’s
706 motion and outputs a classification that can be easily parsed.

707 The chain-of-thought instructions prompt the model to analyze the video in three steps. The first is to
708 ensure the VLM describes the observed motion of the robot and task-relevant objects in detail. The
709 second is to analyze the observed motion and reason about whether the robot is behaving incorrectly
710 or not. The final step is to conclude with a classification in {ok, failure}. Our objective with the
711 runtime monitor is to interfere if the robot is likely to enter a state that is difficult to reset from (e.g.,
712 because it potentially unsafe) or cannot complete the task. We therefore note that the role of the runtime
713 monitor is only to interfere with the operation of the robot if it is clearly failing to complete the task,
714 for example if the robot is behaving in a manner that is potentially unsafe, or if the robot is unlikely to
715 complete the task in the remaining time budget.

716 **Prompt template** We include our prompt template below.

717 "I am the runtime monitor for an autonomous mobile manipulator capable of


```
solving common household tasks. These are frames of the video recorded by
the manipulator's camera system at approximately 1Hz as it is executing its
current task. My job is to analyze the observations of the manipulator and
identify whether it is acting incorrectly or unsafely. The robot may take up
to {TIME_LIMIT} seconds to complete the task.\n
```

```
The manipulator's current task is to {TASK}.\n
The current elapsed time is {TIME} seconds.\n
```

```
I will format my output in the following form:\n
[start of output]\n
```

```
Analysis: On this line, I will precisely describe the full motion of the
manipulator and all task-relevant objects from the beginning to the end of
the accompanying video. I will also carefully describe the final state of
each of the task-relevant objects and the manipulator. Then, I will
characterize whether the robot is clearly failing the task or not, for
example by behaving erratically.
```

```
Since the video only represents the progress up to the current timestep and the
robot moves slowly, I will refrain from making a failure classification
unless the robot takes an unsafe action or the robot clearly can not
complete the task in the allotted time. After explicitly noting the amount
of time that has passed in seconds and explicitly comparing it with the time
limit (e.g., x out of {TIME_LIMIT} seconds), I will decide whether the
robot is in-progress, or whether the robot will fail the task or is behaving
erratically.\n
```

```
Overall assessment: {CHOICE: [ok, failure]}\n
[end of output]\n
```

```
Rules:\n
```

1. If you see phrases like {CHOICE: [choice1, choice2]}, it means you should replace the entire phrase with one of the choices listed. For example, replace the entire phrase '{CHOICE: [A, B]}' with 'B' when choosing option B. Do NOT enclose your choice in '{' '}' brackets. If you are not sure about the value, just use your best judgement.\n
3. Do NOT forget to conclude your analysis with an overall assessment. As indicated above with '{CHOICE: [ok, failure]}', your only options for the overall assessment are 'ok', or 'failure'. \n
4. Always start the output with [start of output] and end the output with [end of output]. \n

```
Output:"
```

718

719 The task descriptions we use are as follows:

```
task_descriptions:
  cover: "pick up the white blanket and use it to fully cover the white box, then
  let go of the blanket"
  close: "close the white box by folding in the two smaller white side lids and
  the bigger white back lid. First, the robot should concurrently push both
  side lids up, followed by folding up the back lid with both arms, without
  grasping the lids with the grippers. Then, the robot's arms should back
  away from the box"
```

B Experiment Details

B.1 Environments

We provide additional details on the simulated environments used to test our failure detection approach. These environments vary in terms of the properties of their training distributions and the distribution shifts under which their policies are evaluated. This results in different qualitative modes of policy failure. Specifically, we consider the PushT domain from [1] and two high-dimensional bi-manual mobile manipulation domains. A visualization is provided in Fig. 6.

- **PushT Domain:** The policy is tasked with pushing a planar “T”-shaped object into a goal configuration. A trajectory is considered successful if the overlap between the “T”-shaped object and its goal exceeds 90% within 300 environment steps. The action space is the 2-DoF linear velocity of the end-effector. We generate OOD test scenarios by non-uniformly randomizing the scale and dimensions of the “T”-shaped object beyond the randomizations contained in the policy’s demonstration data. The policy tends to fail by converging to a locally optimal configuration, where the “T” overlaps with its goal but in the wrong orientation. Since the task can be solved in a number of ways, we include this domain to evaluate the performance of various score functions in the presence of action multi-modality. We refer to [1] for the process of generating demonstration data in this domain.
- **Close Box Domain:** The policy is tasked with closing a box with three lids. A trajectory is considered successful if all three lids are closed within 120 environment steps (24 seconds). The action space is the 14-DoF linear + angular velocities and gripper command for the end-effectors of two mobile manipulators. Demonstration data is generated by an oracle policy that sets a series of waypoints for the end-effectors based on the initial state. We generate OOD test scenarios by non-uniformly randomizing the scale of the box beyond the randomizations contained in the policy’s demonstration data. The policy tends to fail erratically when the robots e.g., collide with the box or its lids, however, task progression failures may also occur. This domain is primarily used to evaluate the detection of erratic policy failures in a bi-manual robot system with a high-dimensional action space.
- **Cover Object Domain:** The policy is tasked with covering a rigid object with a cloth. A trajectory is considered successful if over 75% of the object is covered by the cloth within 80 environment steps (16 seconds). The action space and process of generating demonstration data is identical to that of **Close Box**. We generate OOD test scenarios by non-uniformly randomizing the position of the object beyond the randomizations contained in the policy’s demonstration data. The policy tends to fail by releasing the cover before reaching the object, i.e., placing it on the ground. This domain is used to evaluate the detection of task progression failures, where contextual reasoning over longer durations is required to assess task progress.

B.2 Baselines

We outline the implementation details of our baselines as introduced in §5. First, with the exception of the VLM runtime monitors, all evaluated failure detection methods consist of computing a score $S(\cdot)$ at each policy inference timestep, taking the cumulative sum of scores up to the current timestep t , and then checking if the cumulative sum exceeds a calibrated threshold to detect policy failure. As such, the baselines differ in their *score function*, i.e., how they compute the per-timestep scores that are then summed and thresholded. Intuitively, a good score function should be well-correlated with policy failure, that is, it should output small scores when the policy is succeeding and large ones when it is failing. For example, Fig. 3 demonstrates this property for our proposed temporal consistency detector. We baseline against an extensive suite of score functions, some of which we newly introduce for the case of generative policies, and others that are common in the OOD detection literature [4].

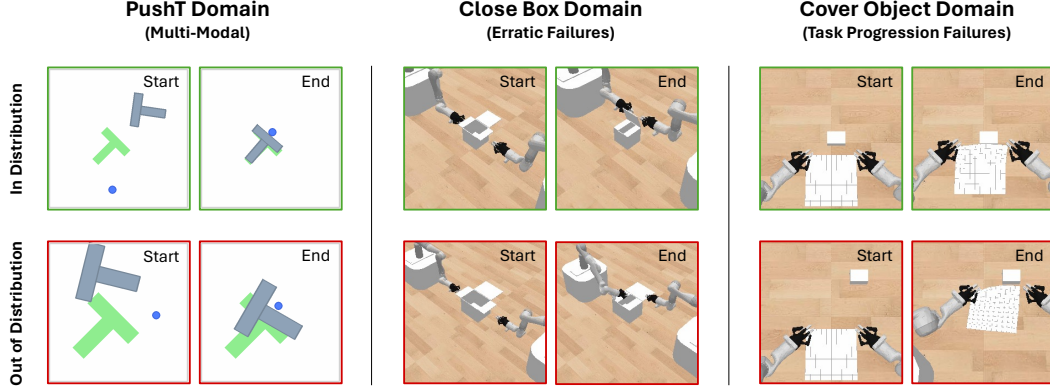


Figure 6: **Task suite.** We evaluate our failure detection approach across three simulated domains. These domains provide coverage over different data distributions (e.g., multi-modality, high-dimensional action spaces) and modes of generative policy failure. For example, generative policies tend to fail erratically in the **Close Box** domain, but smoothly in the **Cover Object** domain. An effective failure detector should be performant across multiple domains rather than just one.

B.2.1 STAC Baselines (Policy Level)

- **Policy Encoder Embedding** quantifies the dissimilarity of the current point cloud observation o_t w.r.t. to the point clouds in the calibration dataset of M successful policy rollouts $\mathcal{D}_\tau = \{\tau^i\}_{i=1}^M$ (§3) within the embedding space of the policy’s encoder (here, o_t denotes the point cloud input to the policy, including the point cloud at the current and previous timestep). More concretely, let E be the policy’s encoder, $z_t = E(o_t)$ be the current point cloud embedding, and $\mathcal{D}_z = E(\mathcal{D}_\tau)$ be the embeddings of all point clouds contained in the calibration dataset. We compute the per-timestep score as the Mahalanobis distance

$$S(z_t; \mathcal{D}_z) = \sqrt{(z_t - \mu_z)^T \Sigma_{zz}^{-1} (z_t - \mu_z)}, \quad (3)$$

where μ_z is the mean and Σ_{zz} is the covariance of the embeddings in \mathcal{D}_z . At test time, we raise a failure warning if the cumulative score η_t exceeds a calibrated detection threshold γ

$$\eta_t > \gamma, \quad \text{where} \quad \eta_t = \sum_{i=0}^t S(z_i; \mathcal{D}_z).$$

Here, γ is set to the $1 - \delta$ quantile of cumulative scores computed over the calibration trajectories $\{\eta_{|\tau^i|}^i\}_{i=1}^M$, where $\tau^i \in \mathcal{D}_\tau$. Importantly, when computing the calibration scores η^i , we do so in a *leave-trajectory-out* fashion: i.e., for a point cloud $o_t \in \tau^i$ where $\tau^i \in \mathcal{D}_\tau$, we compute the per-timestep score as $S(E(o_t); E(\mathcal{D}_\tau \setminus \tau^i))$. This ensures that the dissimilarity of observation o_t is computed w.r.t. trajectories other than its own, which a) aligns with how scores are computed at test time and b) ensures that calibration scores are not trivially low.

We experimented with alternatives to the Mahalanobis distance in Eq. (3), substituting it with top- k scoring for $k \in \{1, 5, 10\}$ based on cosine similarity or L2 distance metrics. However, we found the Mahalanobis distance to be most stable. We also evaluated variants of this baseline that compute the dissimilarity of the full policy state s_t (including both the point cloud embedding and the robots’ end-effector poses) but found equivalent performance.

- **CLIP Pretrained Embedding** quantifies the dissimilarity of the current image observation I_t w.r.t. to the images in the calibration dataset $\mathcal{D}_\tau = \{\tau^i\}_{i=1}^M$ within the embedding space of a pretrained CLIP encoder [66]. The score function (Eq. 3) and calibration process are identical to **Policy Encoder Embedding**. Importantly, the encoder used here is trained with a representation learning objective, which results in a structured embedding space and more interpretable embedding similarity scores. In our experiments, we use the open-source clip-vit-base-patch32 version of CLIP without any fine tuning.

- **ResNet Pretrained Embedding** is identical to **CLIP Pretrained Embedding**, except quantifies image-space dissimilarity using embeddings from a ResNet18 pretrained model [67].
- **Temporal Minimum** is similar to STAC (§A.1) in that it seeks to compute a consistency score between overlapping actions $a_{t+k:t+h|t}$ and $a_{t+k:t+h|t+k}$ sampled from the generative policy at contiguous policy-inference timesteps t and $t+k$, respectively. However, it does so by using a non-statistical distance function. In particular, this baseline computes the per-timestep temporal consistency score at timestep $t+k$ as

$$S(s_{t+k}) = \min_{b \in \{1..B\}} \|a_{t+k:t+h|t} - a_{t+k:t+h|t+k}^b\|, \quad \text{where } a_{t+k:t+h|t+k}^b \sim \tilde{\pi}_{t+k}(\cdot | s_{t+k}).$$

That is, we sample a batch of B actions sequences at timestep $t+k$, compute their L2 distances *w.r.t.* the overlapping actions of the previously executed action sequence $a_{t+k:t+h|t}$, and return the L2 distance associated with the most similar action sequence. Intuitively, this baseline attempts to find the closest action sequence at timestep $t+k$ to previously executed action sequence, in contrast to STAC, which quantifies how well the action distribution $\tilde{\pi}_{t+k}$ at timestep $t+k$ is represented in the distribution $\tilde{\pi}_t$ at timestep t . The values of B are in Table 2. The calibration and runtime procedures of this baseline are identical to STAC (§4.1).

- **Diffusion Reconstruction** adapts the diffusion-based OOD detection approach of Graham et al. [62] for the case of diffusion policy. Specifically, this baseline computes the reconstruction error on re-noised action sequences sampled from the diffusion policy as

$$S(s_t) = \mathbb{E}_{a^0 \sim \pi(\cdot | s_t), \epsilon^i, i} \left[\|a^0 - \epsilon_{\theta}^{i:0}(\sqrt{\bar{\alpha}_i} a^0 + \sqrt{1 - \bar{\alpha}_i} \epsilon^i, s_t)\|^2 \right], \quad (4)$$

where $\epsilon_{\theta}^{i:0}$ denotes the reverse diffusion process from the i -th denoising iteration to the 0-th iteration, resulting in the reconstructed action. We approximate the expectation in Eq. 4 over a batch of $B = 256$ action sequences sampled from the diffusion policy, each re-noised for $i \in \{5, 10, 25, 50\}$ forward diffusion steps (also referred to as *reconstruction depths*). We experimented with several sets of reconstruction depths and found comparable performance. We note that this baseline comes with significant computational expense as it needs to perform the denoising process multiple times: i.e., if we would like to compute R reconstructions, this baseline is approximately R times more expensive than simply sampling the diffusion policy. The calibration and runtime procedures of this baseline are identical to that of STAC (§4.1).

- **Temporal Diffusion Reconstruction** is a temporal variant of **Diffusion Reconstruction** that also computes the reconstruction error on re-noised action sequences sampled from the diffusion policy, but reconstructs the action sequences conditioned on the previous state s_t as

$$S(s_t, s_{t+k}) = \mathbb{E}_{a_{t+k:t+h|t+k}^0 \sim \tilde{\pi}_{t+k}, \epsilon^i, i} \left[\|\hat{a}^0 - \epsilon_{\theta}^{i:0}(\sqrt{\bar{\alpha}_i} \hat{a}^0 + \sqrt{1 - \bar{\alpha}_i} \epsilon^i, s_t)\|^2 \right].$$

Here, \hat{a}^0 denotes the action sequence on which reconstructions are computed, concatenating the first k (executed) actions sampled at timestep t with following $k-h$ (predicted) actions sampled at timestep $t+k$: that is, $\hat{a}^0 = a_{t:t+k|t} \oplus a_{t+k:t+h|t+k}^0$. This step is necessary to ensure that the denoising process conditioned on s_t only considers actions within the policy’s prediction horizon. This baseline represents an alternative form of temporal consistency. Intuitively, it asks whether action sequences sampled at timestep $t+k$ would also be sampled at timestep t , to which the answer is likely yes if the policy is in-distribution, and likely no if the policy is OOD—because the marginal distributions conditioned on s_t versus s_{t+k} may be different. The hyperparameters of this baseline follow **Diffusion Reconstruction**.

- **DDPM Loss** computes the empirical DDPM loss on re-noised action sequences sampled from the diffusion policy as

$$S(s_t) = \mathbb{E}_{a^0 \sim \pi(\cdot | s_t), \epsilon^i, i} \left[\|\epsilon^i - \epsilon_{\theta}(\sqrt{\bar{\alpha}_i} a^0 + \sqrt{1 - \bar{\alpha}_i} \epsilon^i, s_t, i)\|^2 \right].$$

Here, the expectation is taken over a batch of $B = 256$ sampled action sequences and 10 sampled denoising iterations $i \sim \mathcal{U}[0, N)$, where N is the total number of denoising iterations

(§B.3.1). We can think of this baseline as a more efficient version of **Diffusion Reconstruction**, since it directly quantifies the diffusion policy’s performance on its training task without the need to reconstruct actions over numerous denoising iterations. The calibration and runtime procedures of this baseline are identical to that of STAC (§4.1).

- **Temporal DDPM Loss** is a temporal variant of **DDPM Loss** that also computes the empirical DDPM loss on re-noised action sequences sampled from the diffusion policy, but does so conditioned on the previous state s_t as

$$S(s_t, s_{t+k}) = \mathbb{E}_{a_{t+k:t+h|t+k}^0 \sim \tilde{\pi}_{t+k}, \epsilon^i, i} \left[\left\| \epsilon^i - \epsilon_\theta(\sqrt{\bar{\alpha}_i} \hat{a}^0 + \sqrt{1 - \bar{\alpha}_i} \epsilon^i, s_t, i) \right\|^2 \right],$$

where $\hat{a}^0 = a_{t:t+k|t} \oplus a_{t+k:t+h|t+k}^0$ (as defined in **Temporal Diffusion Reconstruction**). The hyperparameters of this baseline follow **DDPM Loss**, over which it is expected to offer advantages via temporal consistency.

- **Diffusion Output Variance** computes the variance B action sequences sampled from the diffusion policy and thresholds it *w.r.t.* the $1 - \delta$ quantile of sample variances computed over the calibration dataset \mathcal{D}_τ . This baseline reflects an alternative output metric to temporal consistency that can be monitored to detect policy failure. While computing output variances might bear resemblance to ensemble methods [68], we note that this approach does not quantify epistemic model uncertainty. Doing so would require training multiple diffusion policies and performing inference with each at test time, which we avoid due to computational expense. The hyperparameters of this baseline are identical to STAC (see Table 2).

Key Characteristics of Baselines First, we highlight that the embedding-based approaches predict failure solely based on the dissimilarity or atypicality of the current state. Hence, these baselines are not *policy aware*: they may raise failure warnings for states that are dissimilar from those contained in the calibration dataset \mathcal{D}_τ without understanding how the policy behaves in those states. In some cases, the policy may still succeed or generalize to minor distribution shifts in state, for which the performance of these baselines will significantly diminish. The reconstruction-based approaches may account for the generalization characteristics of the policy but come with computational expense, which may prohibit their use in real-time settings. The DDPM loss approaches present the next best alternative to our proposed temporal consistency detector, as its score function coincides with the diffusion policy’s training task and do so at negligible computational cost. **Importantly**, we note that the DDPM loss baseline is specific to diffusion policies, whereas STAC is agnostic to the generative policy formulation.

B.2.2 VLM Baselines (Task Level)

As described in §4.2 and §A.2, our primary method for monitoring the robot using the VLM is to prompt the model to analyze a video of the robot’s current task progress in a zero-shot fashion. We contrast the VLMs performance with two baseline variations of this approach.

- **GPT-4o Image:** We introduce a VLM baseline that performs the monitoring task based on a single image rather than the full video of the current progress. The goal of this baseline is to identify the value of video-based VLM reasoning compared to single images. We implement this baseline by querying the VLM using only I_t , the image recorded at the current timestep t , rather than the full video $I_{0:t}$. We also minimally modified the prompt given in §A.2 to refer to the given observation as the “image recorded at the current timestep” instead of a video.
- **GPT-4o Video Success In-Context:** Our primary VLM methodology queries the model in a zero-shot fashion, using only the video of the current task execution and a text description of the task as a reference. Since we provide STAC and the other baselines with a calibration dataset of in-distribution nominal trajectories wherein the policy succeeded at the task, we also investigate whether providing additional videos of successful task execution as a reference can improve the accuracy of the VLM monitor. To do so, we select a single video of an in-distribution successful rollout and provide it to the VLM in-context together with the video

882 of the current task execution. Since the OpenAI API currently only supports video reasoning
883 by querying the model with a list of images, we cannot directly upload two separate videos
884 and reference their filenames in the prompt. Instead, we combine both videos into a long
885 sequence of images with an all-black frame in the middle to mark the end of the reference
886 video and the start of the test video. We then modify the prompt in §A.2 to the below prompt,
887 which explicitly asks the model to compare the two videos. We also tried a version wherein
888 an example generation describing the successful reference video is included as an in-context
889 example, but did not notice meaningful differences in performance, so we only include the
890 prompt that explicitly compares the videos in this appendix.

891 "I am the runtime monitor for a bimanual autonomous mobile manipulator

capable of solving common household tasks. These are the frames of two videos recorded by the manipulator’s camera system at approximately 1Hz. The two videos are separated by a single fully black frame. The first video is a reference video that shows the robot correctly executing the task within {TIME_LIMIT} seconds. The second video shows the robot executing its current task and the frames contain the image observations up to the current timestep. My job is to compare the first reference video with the video of the current task in progress and determine if the robot is still executing the task, or whether it is acting incorrectly or unsafely and should be stopped. In my analysis, I will keep in mind that the reference video shows a completed successful trial, whereas the current video may be shorter: It only shows the current progress and the robot may take up to {TIME_LIMIT} seconds to complete the task. Therefore, I will only interfere when the robot is clearly failing to complete the task or behaving unsafely.\n

The manipulator’s current task is to {TASK}.\n
The current elapsed time is {TIME} seconds.\n

I will format my output in the following form:\n
[start of output]\n
Analysis:\n

1. On this line, I will note the differences between the reference video and the current task video. \n
2. On this line, I will explicitly note the amount of time that has passed in seconds and compare it with the time limit (e.g., x out of {TIME_LIMIT} seconds).

Then, I will decide whether the robot is in-progress, or whether the robot has failed the task or is behaving erratically. Since the video only represents the progress up to the current timestep and the robot moves slowly and takes time to pre-plan its movements, I will refrain from making a failure classification unless the robot takes an unsafe action or the robot has not completed the task in the allotted time.\n

Overall assessment: {CHOICE: [ok, failure]}\n
[end of output]\n

Rules:\n

1. If you see phrases like {CHOICE: [choice1, choice2]}, it means you should replace the entire phrase with one of the choices listed. For example, replace the entire phrase '{CHOICE: [A, B]}' with 'B' when choosing option B. Do NOT enclose your choice in '{ '}' brackets. If you are not sure about the value, just use your best judgement.\n
3. Do NOT forget to conclude your analysis with an overall assessment. As indicated above with '{CHOICE: [ok, failure]}', your only options for the overall assessment are 'ok' or 'failure'. \n
4. Always start the output with [start of output] and end the output with [end of output]. \n

Output:"

892

893 B.3 Evaluation Protocol

894 B.3.1 Diffusion Policies

895 We train a diffusion policy for each environment, using 200 demonstrations for the **PushT** domain
896 and 50 demonstrations for each of the **Close Box** and **Cover Object** domains. We use a diffusion
897 policy architecture identical to the original paper [1] except for the visual encoder. That is, because
898 we use point cloud inputs for our tasks, we substitute the ResNet-based encoder for a PointNet-based

one: a 4-layer PointNet++ encoder [69] with hidden dimension 128. The output of this encoder is concatenated with the proprioceptive inputs and then fed to the noise prediction network. All diffusion policies are specified to perform $N = 100$ denoising iterations. Unless otherwise specified, we use standard settings for the prediction h and execution horizon k of the diffusion policy (details in Table 2).

B.3.2 Constructing the Calibration Dataset

Calibrating STAC (§A.1) and its baselines (§B.2.1) requires a small dataset of successful policy rollouts $\mathcal{D}_\tau = \{\tau^i\}_{i=1}^M$. These rollouts provide grounding on the nominal behavior of the policy as it operates on in-distribution test cases. This allows us to evaluate the test-time behavior of a potentially failing policy *w.r.t.* its known nominal behavior.

Calibration Data Quality We found it important to ensure the quality of trajectories $\tau^i \in \mathcal{D}_\tau$. Specifically, trajectories in which the policy succeeds but in an undesired or unacceptable manner should not be used for calibration. For example, the policy may solve the **Close Box** task (Fig. 6), but damage the lids of the box in the process. Including such a trajectory in the calibration dataset would define this behavior as *nominal* and degrade the sensitivity of the detectors at test time. Returning to our example, the detectors may not raise a failure warning if the policy damages the box while closing it.

Collecting the Calibration Dataset In practice, such a calibration dataset could be collected during a policy validation phase prior to deployment. For instance, we collect $M = 50$ successful policy rollouts for each simulated domain, manually filtering episodes where the policy succeeded with unacceptable behavior (e.g., with jitter). We hypothesize that the performance of the detectors *w.r.t.* the number of rollouts M is task specific. For example, a smaller calibration dataset may be sufficient for tasks with low variability (i.e., in a single, structured environment), while a larger dataset may be necessary if the policy is to be deployed at scale. We note, however, that increasing the calibration dataset size may be desirable to achieve stronger conformal guarantees on the detector’s FPR (as derived in §D).

Calibrating on Demonstration Data Finally, we experimented with variants of STAC that calibrated on trajectories contained in the policy’s demonstration dataset, in attempt to eliminate the need to collect an additional calibration dataset of policy rollouts. However, doing so led to a significant increase in the detector’s FPR. We attribute this to the well-known covariate shift problem for imitation learned policies [24, 25]. That is, their prediction error increases quadratically on states induced under the policy, causing the detectors’ to mistake successful test-time rollouts for failures.

B.3.3 Testing & Evaluation

Instead of evaluating the failure detectors online during policy rollouts, we collect several test datasets of policy rollouts, which may consists of both successes and failures. Each trajectory is labeled either success or failure by thresholding the return at the final state of the episode (as per the definition in §3). We then perform offline evaluation of the failure detectors by invoking them at each timestep of the trajectory, which allows us to identify the first timestep at which the detectors issue a warning.

B.3.4 Definitions & Reported Metrics

We expand on the definitions and metrics described in §3 and §5. We first define a *positive* as a trajectory where the policy fails and a *negative* as trajectory where the policy succeeds. A true positive is counted if the failure detector raises a warning at any timestep in a trajectory where the policy fails. A true negative is counted if the failure detector never raises a warning in a trajectory where the policy succeeds. A false positive is counted if the failure detector raises a warning at any timestep in a trajectory where the policy succeeds. A false negative is counted if the failure detector never raises a warning in a trajectory

where the policy fails. Detection time is defined as the earliest timestep in which the failure detector raises a warning in a trajectory where the policy fails.

In our experiments, we report true positive rate (TPR), true negative rate (TNR), false positive rate (FPR), detection time (DT), accuracy, and balanced accuracy. TPR, also referred to as *sensitivity*, measures the number of true positives (detected failures) over total number of positives (failures). TNR, also referred to as *specificity*, measures the number of true negatives (detected successes) over total number of negatives (successes). FPR measures the number of false positives (incorrectly detected failures) over the total number of negatives (successes). Accuracy and balanced accuracy account for both the TPR and TNR of the detector. However, we report balanced accuracy when test set contains a non-negligible imbalance of positives and negative trajectories.

C Additional Results

C.1 PushT Ablation

We conduct an ablation study on the **PushT** domain to test how the performance of STAC varies with respect to the prediction horizon h and execution horizon k of the diffusion policy. Together, the prediction and execution horizons determine the number of temporally overlapping action components (i.e., between $a_{t+k:t+h|t}$ and $a_{t+k:t+h|t+k}$) that are statistically compared by STAC, while the execution horizon governs how far apart in time the action distributions $\bar{\pi}_t$ and $\tilde{\pi}_{t+k}$ are generated.

The result is shown in Fig. 7. We find that STAC (MMD) performs comparatively across execution horizons of $k = 4$ and $k = 8$, but performs best with the standard diffusion policy settings of $k = 8$ and $h = 16$ (used for the main result in Fig. 4). The detector’s performance drops when using the smallest execution horizon of $k = 2$. We attribute this to the relatively small amount of environment change that occurs within two execution steps, which causes $\bar{\pi}_t$ and $\tilde{\pi}_{t+k}$ be similarly distributed and leads to overly conservative statistical distances. This is corroborated by our findings, where the detectors attain $> 95\%$ TNRs across various execution horizons, but using $k = 2$ leads to a significant drop in TPR to 61%. In comparison, $k = 4$ and $k = 8$ attain TPRs of 78% and 95%, respectively. Overall, STAC’s performance may vary with the selection of the policy’s execution horizon, but is relatively stable across choices of the policy’s prediction horizon.

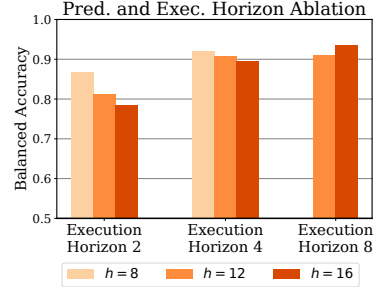


Figure 7: **PushT** ablation on the performance of STAC subject to varying policy prediction and execution horizon settings (Table 2).

C.2 Vision-Language Model Ablation

In this section, we include an additional ablation of our VLM method, as well as an extended discussion on the performance of the VLM methods in §5.

First, we present an additional ablation, augmenting the VLM methods presented in Table 1 with the **GPT-4o Video Success In-Context** baseline presented in §B.2.2. Table 3 shows that providing the VLM with an in-context video of a successful task execution does not significantly change the performance of the VLM reasoner as compared to the zero-shot video approach. While it is intuitive to expect that providing a reference video would facilitate describing the motion of the robot, it is difficult to identify why providing a reference video resulted in negligible changes in performance. Qualitatively, we found the in-context video approach seems to describe the motion of the robot in the video more clearly than zero-shot. However, the successes on the **Close Box** task under distribution shift look different from the nominal successes: Even if the robot succeeds under distribution shift, the robot moves slower than normally. This visual difference in speed leads to some false positives, so

that the overall performance is roughly the same as the zero-shot method. Another possibility is that analyzing the difference between videos is a more challenging task for a VLM compared to describing a single video, or that the manner in which we provide the reference video to the VLM (see §B.2.2) complicates the task.

Second, we discuss the poor performance of the **GPT-4o Single Image** baseline. As shown in Table 1 and Table 3, the single image baseline suffers a 100% false positive rate on the **Close Box** domain. It performs poorly on this task for a simple reason: Without observing the initial state of the box with its lids in an opened position, and without observing the motion of the robot closing the lids, the VLM is unable to clearly identify the lids of the box and whether the box is closed. Therefore, once the time limit to complete the task is exceeded, the monitor declares a failure. We iterated on several prompts that asked more detailed questions about the location of the box and lids in an attempt to coerce this baseline to identify the lids, but to no avail. As a result, all the outputs of the single image baseline resemble the following example of a false positive:

```
"[start of output]
Analysis: The current observation shows the manipulator's arms positioned near
the white box, with the grippers open and not grasping the lids. The two
smaller white side lids and the bigger white back lid of the box are not
visible, suggesting they are not yet folded. The elapsed time is 30 out of
30 seconds, which means the robot has reached the time limit for completing
the task. Given that the lids are not folded and the task is not completed,
the robot is clearly failing the task.

Overall assessment: failure
[end of output]"
```

Overall, we consider these results clear evidence that video-based reasoning offers strict improvements over single image-based reasoning for the assessment of task progression failures.

Failure Detector	Close Box: In-Distribution			Close Box: Out-of-Distribution			Close Box: Combined		
	TPR ↑	TNR ↑	Det. Time (s) ↓	TPR ↑	TNR ↑	Det. Time (s) ↓	TPR ↑	TNR ↑	Accuracy ↑
VLM GPT-4o Image	1.00	0.00	24.00	1.00	0.00	24.00	1.00	0.00	0.32
GPT-4o Video (Ours)	0.60	1.00	22.40	0.52	0.75	21.42	0.53	0.94	0.81
GPT-4o Video + IC success	0.60	1.00	24.00	0.48	0.75	24.00	0.48	0.94	0.80

Table 3: Comparison of VLM runtime monitors on the **Close Box** domain. Note that the GPT-4o Video + IC Success is the only additional baseline that is not already included in Table 1.

C.3 Extended Discussion

Vision-Language Model Analysis While Fig. 5 shows that the VLM provides accurate failure assessments on distribution shifts that result in smooth task-progression failures, it still does not achieve 100% accuracy. Table 1 shows that the VLM struggles to identify erratic policy failures, but well-equipped to detect policy successes, as evidenced by its 94% overall TNR. That is, while the TNR is only 75% on the **Close Box** out-of-distribution split, this only corresponds to 4 problematic false positive episodes (because the robot almost always fails in the out-of-distribution split). Therefore, in this section, we seek to develop an intuition for the reasons why the VLM may miss a detection.

Firstly, a key difference between the erratic failures and the task progression failures is that the erratic failures are often more visually subtle, and hence more challenging to interpret. For example, erratic failures include cases where the robot collides with the box, or moves the lids but does not fully close them. In contrast, the robot takes more obviously wrong actions (e.g., stalling, clearly misplacing the cover) in the task progression failures (see Fig. 5).

Failure Detector		Cover Object: Out-of-Distribution			Close Box: Out-of-Distribution			Combined: Cover + Close		
		TPR \uparrow	TNR \uparrow	Det. Time (s) \downarrow	TPR \uparrow	TNR \uparrow	Det. Time (s) \downarrow	TPR \uparrow	TNR \uparrow	Accuracy \uparrow
Ours	STAC + GPT-4o Video (Full)	0.45	0.92	6.22	0.98	0.60	12.99	0.81	0.83	0.81
	GPT-4o Video	0.35	1.00	7.10	0.91	0.80	20.68	0.73	0.94	0.78
	STAC MMD	0.35	0.92	6.29	0.58	0.80	9.02	0.51	0.89	0.59
	STAC Rev. KL	0.35	0.77	6.51	0.58	0.80	9.89	0.51	0.78	0.57
	STAC For. KL	0.35	0.92	6.40	0.58	0.80	8.00	0.51	0.89	0.59
Embed.	Policy Encoder	1.00	0.69	3.28	0.98	0.20	2.27	0.98	0.56	0.89
	CLIP Pretrained	0.20	0.92	8.80	1.00	0.40	10.53	0.75	0.78	0.75
	ResNet Pretrained	0.05	1.00	8.80	1.00	0.40	16.22	0.70	0.83	0.73
Diffusion	Temporal Min.	0.20	0.77	5.60	0.63	0.80	6.84	0.49	0.78	0.56
	Diffusion Recon. [62]	0.00	0.92	-	0.35	1.00	9.81	0.24	0.94	0.40
	Temporal Diffusion Recon.	0.30	0.92	5.73	0.56	0.80	10.27	0.48	0.89	0.57
	DDPM Loss (Eq. (1))	0.50	0.85	6.32	0.95	1.00	10.60	0.81	0.89	0.83
	Temporal DDPM Loss	0.30	0.92	5.73	0.63	0.80	8.15	0.52	0.89	0.60
	Diffusion Output Variance	0.00	0.92	-	0.23	1.00	10.64	0.16	0.94	0.33

Table 4: **Detecting task progression failures** in the Cover Object and Close Box domains. Our full approach, which combines STAC with the VLM runtime monitor, achieves performance on-par with the top performing baselines for the detection of task progression failures. Importantly, baselines that perform well on this domain, such as Policy Encoder and DDPM Loss, do not maintain consistent performance across the multi-modal (Fig. 4) or erratic failure (Table 1) domains, in contrast to STAC, which achieves both high TPRs and TNRs on these domains. Thereby, the VLM runtime monitor is essential to complementing STAC beyond the detection of erratic failures, resulting in the best overall detector.

Secondly, we can examine the chain-of-thought generation of the VLM to develop an intuition for the VLMs failure modes: We prompt the VLM to first describe the motion of the robot, after which we ask the VLM to assess whether the described motion constitutes nominal execution or a failure. Therefore, we manually examine the VLM outputs for all episodes in which it incorrectly classified an episode as nominal or as a failure, and then attribute the failure to either errors in a) the VLMs description of the video or b) its reasoning as to whether the video constituted nominal behavior or failure.

The result shown in Table 5 indicate that the vast majority of mistakes made by the VLM are caused by the model failing to correctly describe the video. In part, we can attribute this to a visual domain gap between our simulation-based evaluations and the models’ real-world training data: Deficiencies in the simulator’s fidelity may make it difficult for the VLM to correctly identify the cover and the lids of the box. This is mirrored by a similar result in [44], where the authors found open-source VLMs to perform poorly at interpreting images from a self-driving simulator.

Domain	FPR	FNR
Close Box	100%	75%
Cover Object	-	85%

Table 5: **Error attribution of VLM errors.** For each domain, we tabulate what percentage of erroneous VLM classifications result from the VLM describing the video in a way that fundamentally mischaracterizes the motion of the robot. FPR denotes the percentage of false positives: cases in which the VLM incorrectly raises an alarm, caused by visual understanding errors. FNR denotes the percentage of false negatives: cases in which the VLM deems a failure episode as a success. Note that the VLM achieved a 100% FPR on the **Close Box** domain, indicating the all false positives in this domain resulted from the VLM misunderstanding the robot’s motion.

Finally, we remark that our evaluations primarily use GPT-4o. In early experimentation, we compared the use of both GPT-4o and GPT-4-turbo and found they performed similarly. Therefore, we ran the full evaluations with GPT-4o only, as GPT-4o is significantly faster and cheaper than GPT-4-turbo.

Task Progression Failure Analysis We evaluate our proposed approach and the baselines on domains where the policies exhibit task progression failures. The result is shown in Table 4.

First, we observe that the combination of STAC and GPT-4o results in a better overall accuracy than using STAC alone (coinciding with Fig. 5). Upon further analysis, we find that two baselines perform

comparatively. The first is Policy Encoder, which tends to raise a failure warning when the point cloud state deviates from those in the calibration dataset. This strategy performs well because almost all distribution shifts result in policy failure in this domain. However, it incurs a 100% FPR on the erratic failure domain (Table 1), where the policy may succeed under distribution shifts. The DDPM Loss baseline also achieves appreciable performance here, however, it experiences notable performance drops relative to STAC on the multi-modal domain (Fig. 4). These findings highlight the value of our proposed failure categorization. Specifically, because STAC is best able to detect erratic failures among all baselines and the VLM monitor can amicably capture task progression failures (whilst both maintain high TNRs), their combination yields the best overall detector. Moreover, we expect our failure detector’s performance to further improve with the development of more capable VLMs that, for example, may contend with the current shortcomings revealed by our VLM analysis above.

D Derivations

To validate our design choices, we show in this section that the STAC score function and the calibration procedure in §4.1 provably result in a low false positive rate. To do so, we apply recently popularized tools from conformal prediction because they are sample efficient and distribution-free, meaning that they do not require distributional assumptions on the trajectory rollouts. Our guarantee is a direct application of the standard results in split conformal prediction [54], but to ensure the self-containedness of this manuscript, we first briefly reintroduce the core concepts in conformal prediction (taken from [54]) using the notation in our paper.

Background on Conformal Inference In its most basic form, the goal of conformal prediction is to construct a prediction set \mathcal{C} that will contain the true value of a new test point X_{test} with a user defined probability of at least $1 - \delta$ [54]. To do so, a conformal algorithm requires a sequence of calibration samples $\{X^i\}_{i=1}^M$ with all samples $X^1, \dots, X^M, X_{\text{test}}$ i.i.d., as well as a conformity score function $\eta(X) \in \mathbb{R}$. Intuitively, conformal methods use $\{\eta(X^i)\}_{i=1}^M$ to identify how likely $\eta(X_{\text{test}})$ is to lie within the range of a $1 - \delta$ fraction of the calibration samples (i.e., how well X_{test} conforms to the calibration data). We emphasize that this approach ensures that we construct a valid prediction set \mathcal{C} , regardless of the choice of conformity score and without knowing any properties of the data generating distribution:

Theorem 1 (Adapted from Thm. D.1 in [54]). *Let $\mathcal{D}_{\text{calib}} = \{X^1, \dots, X^M\}$ be a calibration dataset and let X_{test} be a test sample. Suppose that the samples in $\mathcal{D}_{\text{calib}}$ and X_{test} are independent and identically distributed (i.i.d.). Then, defining*

$$\gamma := \inf \left\{ \xi \in \mathbb{R} : \frac{|\{i : \eta(X^i) \leq \xi\}|}{M} \geq \frac{\lceil (M+1)(1-\delta) \rceil}{M} \right\}$$

as the $\frac{\lceil (M+1)(1-\delta) \rceil}{M}$ empirical quantile of the calibration data ensures that

$$\mathbb{P}(\eta(X_{\text{test}}) \leq \gamma) \geq 1 - \delta.$$

Here, $\lceil \cdot \rceil$ denotes the ceiling function.

Conformal guarantee of STAC The base split conformal procedure outlined by Theorem 1 requires that the samples used for calibration and test are i.i.d. This is not the case for states and actions observed sequentially within a trajectory, complicating the analysis of applying the STAC detector at each timestep within a trajectory. To solve this issue and provide a guarantee when we sequentially apply our detector on the correlated state action pairs within a trajectory, we calibrate the detector using the consistency scores generated across trajectories in §4.1. This allows us to rigorously bound the false positive rate using Theorem 1.

Corollary 1 (STAC has low FPR). *Let $\mathcal{D}_{\tau} = \{\tau^i\}_{i=1}^M \stackrel{\text{iid}}{\sim} P_{\tau}$ be the validation dataset of successful trajectories, each consisting of H_i timesteps and drawn i.i.d. from the closed-loop nominal distribution*

1075 P_τ . Moreover, let η_t be defined as the STAC temporal consistency score at some timestep $t \geq 0$ in
 1076 equation Eq. (2) and set γ equal to the empirical $\frac{\lceil (M+1)(1-\delta) \rceil}{M}$ quantile of the terminal STAC scores
 1077 $\{\eta_{H_i}^i\}_{i=1}^M$ of the trajectories in \mathcal{D}_τ . Then, the false positive rate, that is, the probability that we raise a
 1078 false alarm at any point during a new successful test trajectory $\tau \sim P_\tau$, is at most δ . I.e.,

$$\text{FPR} := \mathbb{P}_{P_\tau}(\exists t \geq 0 \text{ s.t. } \eta_t > \gamma) \leq \delta. \quad (5)$$

1079 *Proof.* Let H be the length of the test trajectory τ . If there is no distribution shift, i.e., when the test
 1080 trajectory τ is i.i.d. with respect to $\mathcal{D}_\tau \stackrel{\text{iid}}{\sim} P_\tau$, it holds that η_H and $\{\eta_{H_i}^i\}_{i=1}^M$ are i.i.d. Therefore, by
 1081 Theorem 1, we have that

$$\mathbb{P}_{P_\tau}(\eta_H > \gamma) \leq \delta.$$

1082 Moreover, since we define $\eta_t = \sum_{i=0}^{j-1} \hat{D}(\bar{\pi}_{ik}, \bar{\pi}_{(i+1)k})$ for $t = jk$ in Eq. (2) and since $\hat{D}(\cdot, \cdot) \geq 0$ because
 1083 it is a statistical distance, it follows that η_t is increasing. That is, $\eta_0 \leq \eta_k \leq \eta_{2k} \leq \dots \leq \eta_H$. Therefore, if
 1084 η_t crosses the threshold γ at any time, it also holds that $\eta_H > \gamma$. This immediately implies the corollary,
 1085 as we then have that

$$\mathbb{P}_{P_\tau}(\exists t \geq 0 \text{ s.t. } \eta_t > \gamma) = \mathbb{P}_{P_\tau}(\eta_H > \gamma) \leq \delta.$$

1086 □

1087 We conclude this section with three remarks:

- 1088 1. We only bound the false positive rate, which ensures that our algorithm does not raise a false
 1089 alarm with high probability, so that any warnings likely correspond to an OOD scenario. We
 1090 do so because a system that raises too many false alarms is impractical to use. Our calibration
 1091 approach does not guarantee the detection of failure modes, nor does it guarantee that we
 1092 do not issue warnings on OOD successes, as this is not possible without any distributional
 1093 assumptions on the OOD scenarios or without using failure data for calibration [70]. Instead,
 1094 we empirically find that our temporal consistency score performs amicably at detecting
 1095 failures in our experiments (§5).
- 1096 2. Corollary 1 only certifies that the false positive rate of the STAC detector is low. We make
 1097 no claims on the overall performance of the combination of the VLM and STAC, as we use
 1098 the VLM as a zero-shot black-box classifier. Future work could investigate methodologies to
 1099 jointly calibrate an ensemble of failure detectors.
- 1100 3. We emphasize that conformal guarantees, like those in Theorem 1 and Corollary 1, are
 1101 *marginal* with respect to the calibration data: They may not hold exactly when given a
 1102 particular calibration dataset, but if we were to sample thousands of calibration datasets, we
 1103 would find our guarantees hold on average. Therefore, as expected, STAC does not exactly
 1104 satisfy Eq. (5) in our experiments (§5), as compute budgets limited our experimentation to
 1105 repetitions on a limited number of random seeds.