

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](https://sites.google.com/view/detecting-policy-failure).

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

## 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  $\pi$  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  $\beta_1$  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  $\beta_2$  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  $\beta_1$ , and KDE bandwidth  $\beta_2$  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  $\beta_1$  and  $\beta_2$ , respectively. For example, setting  $\beta_2$  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 ( $\beta_1$ )	Median Heuristic [64, 65]	$1.0/ \mathcal{A} $
KDE bandwidth ( $\beta_2$ )	$\sqrt{\lambda_{\max}(\text{Cov}(a_{t+k:t+h}))}$	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  $\nu$  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].

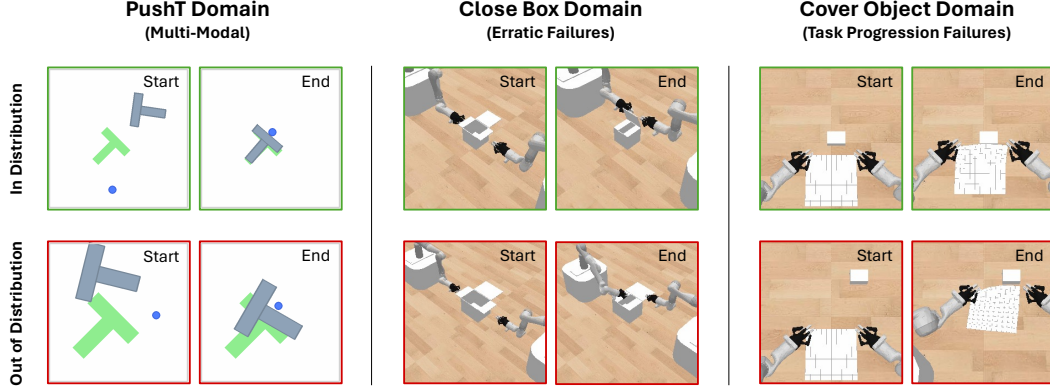


Figure 6: **Task suite.** We evaluate our failure detection approach across three simulated domains. These domains provide coverage over different data distributions (e.g., multi-modality, high-dimensional action spaces) and modes of generative policy failure. For example, generative policies tend to 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.

### 766 B.2.1 STAC Baselines (Policy Level)

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

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

774 where  $\mu_z$  is the mean and  $\Sigma_{zz}$  is the covariance of the embeddings in  $\mathcal{D}_z$ . At test time, we  
775 raise a failure warning if the cumulative score  $\eta_t$  exceeds a calibrated detection threshold  $\gamma$

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

776 Here,  $\gamma$  is set to the  $1 - \delta$  quantile of cumulative scores computed over the calibration  
777 trajectories  $\{\eta_{|\tau^i|}^i\}_{i=1}^M$ , where  $\tau^i \in \mathcal{D}_\tau$ . Importantly, when computing the calibration scores  
778  $\eta^i$ , 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  
779 compute the per-timestep score as  $S(E(o_t); E(\mathcal{D}_\tau \setminus \tau^i))$ . This ensures that the dissimilarity  
780 of observation  $o_t$  is computed w.r.t. trajectories other than its own, which a) aligns with how  
781 scores are computed at test time and b) ensures that calibration scores are not trivially low.

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

- 787 • **CLIP Pretrained Embedding** quantifies the dissimilarity of the current image observation  
788  $I_t$  w.r.t. to the images in the calibration dataset  $\mathcal{D}_\tau = \{\tau^i\}_{i=1}^M$  within the embedding space  
789 of a pretrained CLIP encoder [66]. The score function (Eq. 3) and calibration process are  
790 identical to **Policy Encoder Embedding**. Importantly, the encoder used here is trained  
791 with a representation learning objective, which results in a structured embedding space and  
792 more interpretable embedding similarity scores. In our experiments, we use the open-source  
793 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  $\mathcal{D}_\tau$ . 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  $\mathcal{D}_\tau$  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.

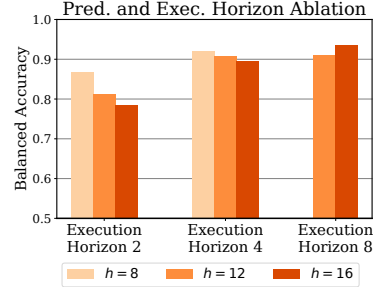


Figure 7: **PushT** ablation on the performance of STAC subject to varying policy prediction and execution horizon settings (Table 2).

### 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  $X_{\text{test}}$  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  $X_{\text{test}}$  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  $X_{\text{test}}$  be a test sample. Suppose that the samples in  $\mathcal{D}_{\text{calib}}$  and  $X_{\text{test}}$  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  $P_\tau$ . Moreover, let  $\eta_t$  be defined as the STAC temporal consistency score at some timestep  $t \geq 0$  in  
 1076 equation Eq. (2) and set  $\gamma$  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  $\mathcal{D}_\tau$ . 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  $\delta$ . 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  $\tau$ . If there is no distribution shift, i.e., when the test  
 1080 trajectory  $\tau$  is i.i.d. with respect to  $\mathcal{D}_\tau \stackrel{\text{iid}}{\sim} P_\tau$ , it holds that  $\eta_H$  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  $\eta_t$  is increasing. That is,  $\eta_0 \leq \eta_k \leq \eta_{2k} \leq \dots \leq \eta_H$ . Therefore, if  
 1084  $\eta_t$  crosses the threshold  $\gamma$  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.