Focus on the Moment: Real-Time Semantic Failure Detection in Vision-Language-Action Models

Seul Kim, Hee Bin Yoo*, Byoung-Tak Zhang*
Seoul National University
*Corresponding author

***Abstract*—Vision-language-action (VLA) policies are prone to *semantic* failures, wherein the motion remains plausible while the policy manipulates on the wrong target. Such failures are recoverable if detected during execution. However, existing detectors require complete trajectories and provide only post hoc assessments, determining *whether* a failure occurred but not *when*. To *focus on the moment*, we propose FailMoment, which detects failure at each frame by utilizing only the recent past. This enables trajectory-level, frame-level, and real-time predictions within a single model under an inference latency budget. As training FailMoment requires frame-level labels, we introduce SemGen, a pipeline that converts successful demonstrations into semantic failures and pinpoints the failure frame from the gripper signal, producing 645,029 labeled frames from 3,148 trajectories across 80 RLBench and LIBERO tasks. FailMoment remains comparable with dedicated whole-trajectory detectors. Specifically, on wrong-position failures in the seen-task split, it achieves a trajectory-level AUROC of 0.945, compared with 0.865 for the strongest baseline. It substantially outperforms baselines in frame-level detection, reaching 0.993 AUROC, while the best baseline reaches only 0.559, and fires on time in 71% of failures, whereas no baseline reaches 8%. Furthermore, we verify our method is transferable to the real world with the RT-1 dataset.**

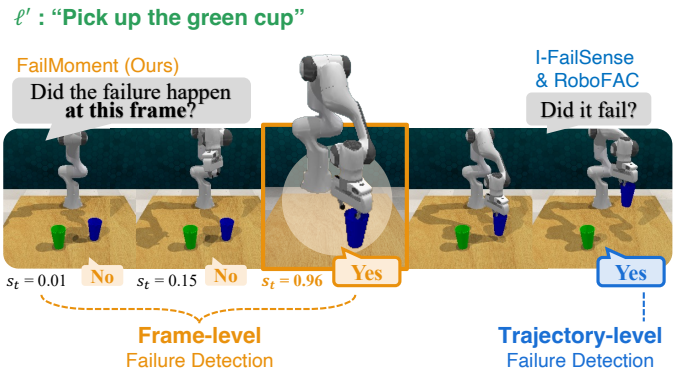


Fig. 1: **Focus on the moment.** Given the instruction ℓ' *"pick up the green cup"*, the policy instead commits to the blue cup. Whole-trajectory detectors judge *whether* a trajectory failed after it ends, whereas **FailMoment** scores each frame with a semantic mismatch score s_t — it stays low during ambiguous motion and rises when the gripper commits to the wrong target.

I. Introduction

A vision-language-action (VLA) policy can execute a physically plausible motion that mismatches the task instruction, producing a *semantic failure*. For example, when instructed to *"pick up the green cup,"* the policy may produce a valid grasping motion but close on the wrong cup (Figure 1). We refer to such errors as *semantic failures*: the motion remains physically plausible, but the object, attribute, or spatial relation being acted upon does not match the instruction. These are relevant in scenes with distractors, where multiple objects are present, and the policy should manipulate the correct object specified by the language instruction [12].

Semantic failures are recoverable if detected during execution. Once the policy commits to the wrong target, a detector can intervene, redirect the policy, or trigger recovery before the task is completed. However, existing semantic failure detectors operate primarily at the trajectory level. They consume an entire rollout and output judgment only after execution ends [6, 7, 21]. This answers *whether* a failure occurs, but not *when*. As a result, they provide post hoc assessments rather than the real-time signal needed for recovery. Instead of judging a completed trajectory, the detector should evaluate each frame and identify when the policy first commits to the wrong target. This frame-level formulation naturally supports real-time detection, allowing the detector to produce a semantic mismatch score at each step as the trajectory unfolds and fire near the moment of failure. The main obstacle is supervision: training such a detector requires temporally localized labels, yet coarse trajectory-level supervision provides insufficient temporal signal for localization [17]. Consequently, existing datasets provide only trajectory-level labels [6, 7, 21], and detectors trained on them inherit the same limitation.

In this work, we reframe semantic failure detection from a question of *"did it fail?"* to temporal localization of *"when did it fail?"*. To this end, we introduce **SemGen**, a pipeline that converts successful demonstrations into semantic failures through counterfactual instruction relabeling, spanning three failure modes: wrong color, object, and position. It applies to both simulated and real-robot trajectories whenever the gripper signal is logged, automatically assigning the failure frame. Across 80 RLBench and LIBERO tasks, it produces 645,029 labeled frames from 3,148 trajectories. Using this data, we train **FailMoment**, a detector that evaluates each frame using only recent context. This design forces the model to *focus on the moment* rather than rely on the full trajectory, supporting trajectory-level, frame-level, and real-time prediction.

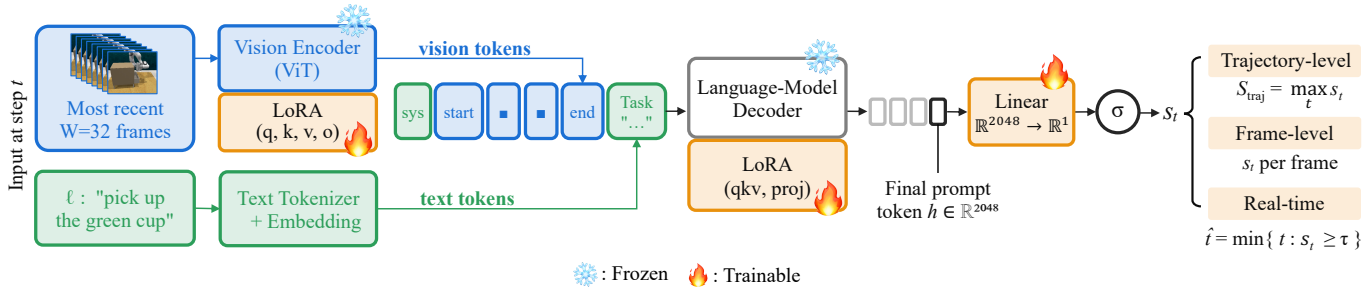


Fig. 2: **FailMoment Architecture. FailMoment** uses a frozen vision encoder and a frozen language-model decoder, both with LoRA. At each timestep t, it encodes the most recent frames and the instruction ℓ, and a linear head maps the hidden state of the final token to the semantic mismatch score s_t. The trajectory-level score is the maximum over the stream, the frame-level output is s_t at each frame, and the real-time alarm fires at the first threshold crossing.

Experimentally, **FailMoment** remains competitive with baselines at the trajectory level, and substantially outperforms them in frame-level localization and on-time detection. Concretely, on wrong-position failures, its trajectory-level AUROC (0.945/0.700 for seen/unseen tasks, respectively) is comparable to the strongest baseline (0.865/0.720). Its frame-level AUROC (0.993/0.986) far exceeds the best baseline (0.559/0.612), and it fires on time on 71%/58% of failures where no baseline reaches 8%, all within the real-time stride budget. Furthermore, fine-tuned on relabeled RT-1 [4] trajectories, it transfers to real-robot data across three detection modes.

Our contributions are summarized as follows:

1. We frame semantic failure detection as a *frame-level* problem, asking not only *whether* a trajectory fails, but also *when* the motion commits to the wrong target.
2. We build **SemGen**, a data pipeline that turns successful demonstrations, simulated or real, into semantic failures across three modes: wrong color, wrong object, and wrong position, yielding the first frame-level semantic failure supervision at scale.
3. We propose **FailMoment**, a semantic failure detector that scores each frame using only recent context, enabling trajectory-level, frame-level, and real-time prediction.
4. We show that **FailMoment** remains competitive with baselines at trajectory-level detection, substantially improves frame-level and on-time detection, and transfers to real-robot data on RT-1.

II. Related Works

A. Semantic failure detection

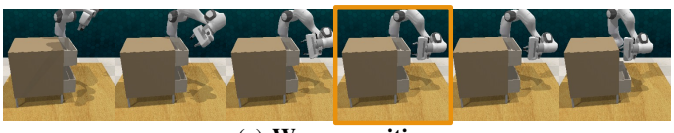
Detectors capable of handling semantic failures evaluate whether an executed trajectory satisfies the language instruction. REFLECT [15] queries a language model over a summary of a completed rollout to explain why it failed. AHA [6] trains a VLM to detect failures and explain their causes, while RoboFAC [21] trains a VLM to analyze robot failures from full videos. Most relevant to our setting, I-FailSense [7] post-trains a VLM to detect semantic mismatch between the instruction and the trajectory. Despite these differences, existing methods operate only after execution. They require the entire trajectory as a structured summary [15], an aggregated keyframe grid [6, 7], or a full video [21]. As a result, they produce a single trajectory-level judgment of *whether* a failure occurred but not *when*, which is too late for recovery. We adopt I-FailSense and RoboFAC as our main baselines, since they take single-camera RGB input, matching our setting, whereas REFLECT uses depth, audio, and proprioceptive state, and AHA relies on a multi-view keyframe grid.

B. Semantic failure data

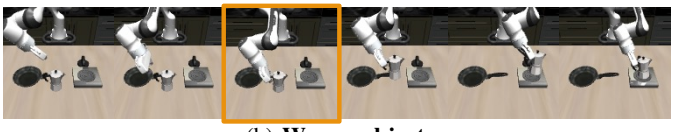
Semantic failure data is typically constructed through one of two strategies: injecting incorrect behavior into the execution or relabeling a successful trajectory with a counterfactual instruction. AHA [6] uses FailGen to procedurally perturb successful demonstrations, including wrong-target-object failures. RoboFAC [21] similarly injects failures into robot executions, including wrong-object failures. By contrast, RoboReward [13] and I-FailSense [7] change the instructions of successful episodes to construct semantic negatives for reward modeling and semantic failure detection. Although these datasets induce failures differently, their supervision remains trajectory-level, indicating only *whether* an execution is inconsistent with the instruction, but not *when*. **SemGen** brings frame-level supervision to semantic failure data in a simple but novel way. It first relabels a successful trajectory with a counterfactual instruction, then uses the logged gripper signal to identify the moment when the unchanged motion commits to the wrong target. This yields frame-level labels for both simulated and real-robot trajectories without additional failure generation or manual annotation.

C. Real-time failure detection

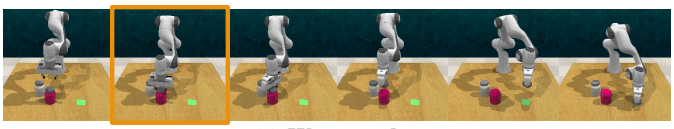
Real-time failure detection seeks to identify failures during execution, but most existing methods focus on behavioral deviations. FAIL-Detect [20] and FIPER [18] score how far the policy deviates from successful behavior, while Sentinel [1] tracks temporal consistency and task progress. SAFE [8], designed for VLAs, uses the policy's hidden features to estimate the likelihood of task failure, but it produces a general failure score rather than identifying when the instruction and the motion semantically diverge. Despite measuring different



(a) **Wrong position.**
*"close the **top** drawer"* → *"close the **bottom** drawer"*.



(b) **Wrong object.**
*"put the **moka pot** on the stove"* → *"put the **frypan** on the stove"*.



(c) **Wrong color.**
*"open the **rose** jar"* → *"open the **white** jar"*.

Fig. 3: **Semantic failures by counterfactual relabeling.** Each row swaps the target in the instruction from the original (ℓ, **blue**) to a distractor in the same scene (ℓ', **orange**), so the unchanged motion now violates ℓ'. The failure frame $t^\star$ (orange box) is read from the gripper signal.

signals, these methods primarily ask whether the motion looks abnormal, not whether it matches the instruction. As a result, a smooth and in-distribution motion toward the wrong target can appear normal, even though it violates the task. In contrast, **FailMoment** checks the trajectory against the instruction frame by frame, enabling real-time detection of semantic failures as they occur.

III. SemGen

SemGen constructs semantic failures whose wrong target and failure frame are both known by design (Figure 3). The pipeline asks only two things of a data source: successful demonstrations to relabel, and a gripper signal with identifiable grasp and release events.

A. Counterfactual instruction relabeling

We start from successful demonstrations with robot state, a logged gripper signal, and scene-object information that specifies the available swap targets. To synthesize a failure, the motion is kept unchanged and the instruction ℓ is rewritten into ℓ' by redirecting its referring expression from the original target to a different target available in the same scene. The same trajectory succeeds under ℓ but acts on the wrong target under ℓ', making the failure semantic by construction and forcing the detector to ground the instruction in the scene rather than read the motion alone.

B. Semantic failure modes

Figure 3 shows the three modes of relabeling, each restricted to a target the scene actually contains. *Wrong position* replaces the positional word with another feasible position. For example, *"close the **top** drawer"* becomes *"close the **bottom** drawer"*, chosen only when the bottom drawer is also open. *Wrong object* swaps the object's noun phrase for a different graspable object type in the scene. For example, *"put the **moka pot** on the stove"* becomes *"put the **frypan** on the stove"*, with a frypan available. *Wrong color* keeps the object and changes only its color. For example, *"open the **rose** jar"* becomes *"open the **white** jar"*, with a white jar present.

C. Locating the moment

For a relabeled trajectory, $t^\star$ is the first frame where the motion *commits* to the wrong target, when the mismatch becomes visible. It is automatically identified from grasp and release events in the gripper signal. For instance, in *pick up the cup*, $t^\star$ is the first frame the gripper closes on the wrong cup. In *close the jar*, it is the first frame the held lid touches down on the wrong jar. Per-frame labels follow,

$$y_t \;=\; \mathbb{1}[\,t = t^\star\,], \tag{1}$$

and every frame of a success trajectory is labeled negative.

The advantage of **SemGen** is that it automatically labels *when* each failure occurs from the gripper signal, regardless of whether the demonstration is simulated or real. This frame-level ground truth is what **FailMoment** trains on, and what prior failure datasets cannot supply.

D. Dataset

SemGen constructs a simulated set $\mathcal{S}$ from two complementary simulators, RLBench [11] and LIBERO [14], to cover a diverse range of semantic failures. RLBench parameterizes its tasks by color and position variants, supplying the wrong-color and wrong-position trajectories. LIBERO places many distinct objects together in one scene and supplies most of the wrong-object trajectories. For RLBench, we run the simulator and extract grasp and release events from the recorded gripper state. In LIBERO, we use the shipped demonstrations and recover the corresponding events from the gripper command logged with each trajectory. Across 80 tasks (15 RLBench and 65 LIBERO), $\mathcal{S}$ comprises 1,944 successful trajectories and 3,148 frame-labeled failure trajectories (wrong color 968, wrong object 1,540, wrong position 640). The trajectories are recorded from a single front-view RGB camera at 20Hz and average ≈!205 frames (≈!10s; median 164, range 77–1,143). We split these into 72 seen tasks for training and 8 fully held-out unseen tasks. Appendix A provides the task lists and seen/unseen split, Appendix B qualitative relabeled samples, and Appendix C the full swap vocabulary.

Figure 4 shows where the failure frame lands within each trajectory ($t^\star/T$). The failures span almost the entire trajectory ($t^\star/T$ from 0.077 to 0.973), and the three modes peak at different points. Wrong-position failures surface earliest (median 0.319); wrong-object failures also surface early (median 0.378); wrong-color failures are broadly distributed (median 0.554). Across modes, failures occur well before the end: the median is $t^\star/T = 0.397$, and 75% occur before $t^\star/T = 0.568$. A detector that waits for the full trajectory to finish reports the failure only after the opportunity for recovery has passed.

TABLE I: **Trajectory- and frame-level failure detection.** Per-mode AUROC/AUPRC on seen and unseen task splits for trajectory- and frame-level evaluation. Best and second-best results are bolded and underlined. **FailMoment** and I-FailSense use 3B VLM backbones (Qwen2.5-VL-3B-Instruct and PaliGemma2-3B [19]); RoboFAC uses Qwen2.5-VL-7B.

			Trajectory-level						**Frame-level**					
			seen			unseen			seen			unseen		
Method	Train on our data	Metric	color	object	position	color	object	position	color	object	position	color	object	position
Qwen2.5-VL-3B [2]	✗	AUROC	0.777	0.492	0.382	0.669	0.631	0.203	0.628	0.646	0.559	0.587	0.676	0.612
		AUPRC	0.493	0.236	0.210	0.363	0.304	0.157	0.190	0.552	0.136	0.268	0.277	0.330
GPT-4o [10]	✗	AUROC	0.583	0.555	0.508	**0.795**	0.638	0.571	0.640	0.549	0.506	0.783	0.489	0.468
		AUPRC	0.316	0.277	0.231	0.506	0.330	0.276	0.202	0.459	0.118	0.528	0.204	0.214
I-FailSense [7]	✓	AUROC	**0.940**	0.869	0.865	0.698	**0.839**	**0.720**	0.577	0.472	0.491	0.553	0.490	0.414
		AUPRC	**0.825**	**0.816**	0.794	0.412	**0.750**	**0.515**	0.148	0.413	0.114	0.217	0.194	0.200
RoboFAC [21]	✓	AUROC	0.815	0.734	0.706	0.758	0.573	0.633	0.607	0.519	0.528	0.680	0.449	0.440
		AUPRC	0.497	0.506	0.332	**0.589**	0.479	0.288	0.189	0.449	0.131	0.382	0.163	0.208
FailMoment	✓	AUROC	0.922	**0.956**	**0.945**	0.724	0.788	0.700	**0.935**	**0.948**	**0.993**	**0.977**	**0.922**	**0.986**
		AUPRC	0.755	0.786	**0.824**	0.435	0.507	0.511	**0.558**	**0.892**	**0.954**	**0.909**	**0.726**	**0.946**

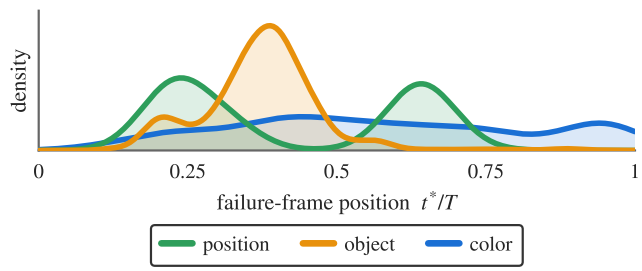


Fig. 4: **Where the failure moment lands.** Density of the normalized failure-frame position $t^\star/T$ across 3,148 failure trajectories by mode. Failures occur before the end, with median $t^\star/T = 0.397$ and 75% before $t^\star/T = 0.568$.

IV. FailMoment

A. Detector

The detector is realized f_θ with a vision-language model (VLM). At each timestep t, f_θ reads the most recent W frames as a short video clip, pairs it with the task instruction ℓ, and emits a semantic mismatch score $s_t \in [0, 1]$. The clip and ℓ form the VLM's standard multimodal prompt (Appendix D), and a single linear layer maps the final-token hidden state to a scalar logit z_t, which is converted to the semantic mismatch score s_t by a sigmoid activation:

$$z_t = f_\theta\big(x_{t-W+1:t}, \ell\big), \qquad s_t = \sigma(z_t). \tag{2}$$

Since each score depends only on the recent window, it detects when the motion commits to a target inconsistent with the instruction. Sliding the window produces a stream of scores, where the frame-level output is s_t itself and the trajectory-level score is the maximum score over the stream,

$$S_{\text{traj}} = \max_t s_t, \tag{3}$$

and the real-time alarm fires at the first threshold crossing,

$$\hat{t} = \min\{\, t : s_t \geq \tau \,\}. \tag{4}$$

B. Training

Fig. 2 shows that **FailMoment** instantiates f_θ on Qwen2.5-VL-3B-Instruct [2] with $W = 32$. Low-Rank Adapters (LoRA) [9] ($r = 32$, $\alpha = 64$, dropout 0.05) are inserted into the self-attention projections of both the language decoder and the vision encoder, covering the query, key, value, and output projections and their vision-encoder counterparts. The adapters and the linear head are the only trainable parameters and are kept in single-precision floating point. The frozen backbone runs in bfloat16 with FlashAttention-2 [5].

Training windows are extracted from $\mathcal{S}$ by sliding a W=32 window at stride 10. Following Eq. 1, each training window inherits the label of its terminal frame; label-1 windows are positive and label-0 windows are negative. For failure trajectories, we follow the real-time scenario in which, once the detector catches the failure moment, control passes to recovery. Accordingly, only the window ending at $t^\star$ is positive, and windows after $t^\star$ are excluded. Success trajectories are slid in full, with every window labeled negative. This yields 77,760 training windows, which are composed of 2,663 positive windows and 75,097 negative windows. During training, positive windows are oversampled and negative windows are undersampled to form 1:1 class-balanced minibatches.

Training objective is an *asymmetric* binary cross-entropy (BCE) loss that assigns a larger weight to negative windows,

$$\mathcal{L} = \frac{1}{N}\sum_i w(y_i)\,\mathrm{BCE}\big(z_i, y_i\big), \quad w(y) = \begin{cases} 1 & y = 1, \\ \beta & y = 0, \end{cases} \tag{5}$$

with $\beta = 3$. During deployment, a failure corresponds to a single moment within a rollout of hundreds of frames. The detector therefore sees a long stream of negatives with at most one positive window at $t^\star$. Setting $\beta > 1$ penalizes false positives more, encouraging the detector to avoid false alarms and fire only with strong evidence. We minimize the loss with AdamW [16] using learning rate 5×10^{-5}, cosine scheduling, 3% warmup, weight decay 0.01, gradient clipping at 1.0, and batch size 8 for 2 epochs (25 h on 2× A100 80GB).

V. Experiment

We conduct the experiments at the three granularities of FailMoment — trajectory-level (Section V-A), frame-level (Section V-B), and on-time detection (Section V-C). The former two experiments are assessed with threshold-free metrics: the area under the receiver operating characteristic curve (AUROC), which measures how well failures are ranked above successes, and the area under the precision-recall curve (AUPRC), which summarizes precision across recall levels. Meanwhile, on-time detection verifies whether the first alarm fires at the exact moment under a fixed threshold, and evaluates per-step latency to ensure real-time computation (Section V-E). Finally, we demonstrate the sim-to-real transfer of our method to a real robot (Section V-F).

As baselines, we use I-FailSense [7] and RoboFAC [21], two state-of-the-art VLM-based semantic failure detectors. For a fair comparison, we fine-tune the baselines using SemGen, following their original training protocols. Thus, the difference of the detection method attributes solely to the performance gap. I-FailSense follows its Stage two recipe (training the K=3 FS blocks with the VLM kept frozen), and RoboFAC follows its LoRA recipe on the failure-detection question type rather than full fine-tuning. We additionally evaluate Qwen2.5-VL-3B [2] and GPT-4o [10] as zero-shot baselines. Qwen2.5-VL-3B isolates the contribution of our supervision. GPT-4o instead probes whether a state-of-the-art model can identify the mismatch without task-specific training.

A. Trajectory-level detection

Trajectory-level detection evaluates whether a trajectory is classified as a failure or a success. For **FailMoment**, Qwen2.5-VL-3B, and GPT-4o, a single trajectory score S_{traj} is obtained by max-pooling the per-frame scores $\{s_t\}$ (Eq. 3). I-FailSense and RoboFAC are originally designed as trajectory-level detectors. I-FailSense takes a single 4-frame grid image sampled across the whole trajectory and outputs a trajectory-level failure score. RoboFAC is prompted to answer whether a semantic failure has occurred, and we take its probability of answering "Yes" as the failure score. This setting tests whether the frame-wise design of **FailMoment** preserves performance in the conventional post-hoc trajectory-level evaluation. The evaluation pool is 300 failure and 300 success trajectories per split, mode-balanced.

Trajectory-level columns of Table I show that I-FailSense leads seen wrong-color (0.940 AUROC) and transfers to unseen object and position (0.839, 0.720), while GPT-4o leads unseen wrong-color (0.795). **FailMoment** is on par with these dedicated detectors, reaching 0.922–0.956 AUROC across the three modes on seen tasks and, on unseen tasks, achieving 91–97% of the best baseline performance in each mode. In AUPRC, **FailMoment** achieves the best seen wrong-position score (0.824) and remains second-best on unseen wrong-object and wrong-position failures (0.507 and 0.511).

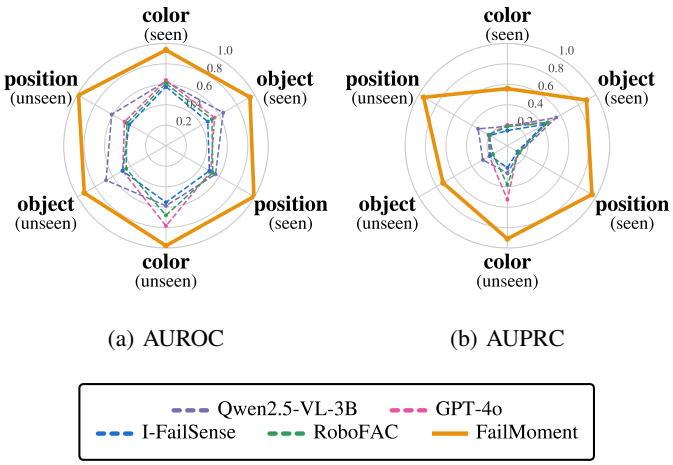


Fig. 5: **Frame-level failure detection.** Per-mode (a) AUROC and (b) AUPRC, each a radar over six cells, the three failure modes (color / object / position) on the seen and unseen task splits. **FailMoment** (solid) sits near the rim on all twelve cells, while the others (dashed) collapse toward the 0.5 chance level.

B. Frame-level detection

Frame-level detection asks whether each window is correctly classified as failure or success, with score s_t assigned to its terminal frame t. Within each mode, the window at the failure frame $t^\star$ is the single positive, and earlier windows ($t < t^\star$) from the same trajectories are negatives. Windows after $t^\star$ are excluded, matching training. Since I-FailSense and RoboFAC do not produce per-frame scores natively, we run them over sliding windows: at each frame t, the recent $W = 32$ frames are fed in each model's native format, a 4-frame grid for I-FailSense and a video clip for RoboFAC. The zero-shot Qwen2.5-VL-3B backbone is evaluated like **FailMoment**, using one $P(\text{"Yes"})$ per window. The evaluation pool contains 3,248 seen and 2,985 unseen windows from held-out trajectories.

Figure 5 and the frame-level columns of Table I show that **FailMoment** identifies the failure frame at 0.922–0.993 AUROC and 0.558–0.954 AUPRC across all modes on both seen and unseen tasks. In contrast, the baselines remain near chance. GPT-4o reaches 0.783 AUROC on unseen wrong-color but only 0.468–0.646 on the other five settings. Among I-FailSense, RoboFAC, and the zero-shot Qwen2.5-VL-3B backbone, the best single result is RoboFAC at 0.680 AUROC.

C. On-time detection

On-time detection measures whether a method fires at the right *moment* (Figure 6). We count a failure trajectory as on-time when its first-fire frame $\hat{t}$ lands in the *on-time zone*, the stride bin of width $S = 10$ containing the failure frame $t^\star$:

$$h \;=\; \mathbb{1}\big[\hat{t} \in [\lfloor t^\star/S \rfloor S,\ \lfloor t^\star/S \rfloor S + S]\big]. \tag{6}$$

For example, a failure at $t^\star$=62 with S=10 falls in the $[60, 70]$ bin, and a first fire at frame 60 or 70 counts as on-time.

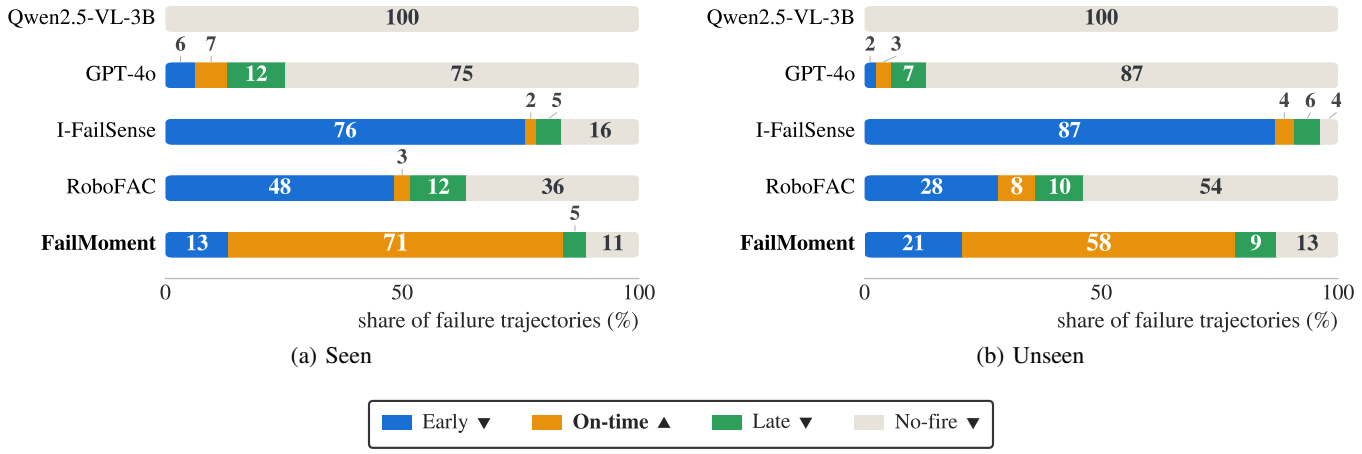


Fig. 6: **On-time detection breakdown** ($\tau = 0.7$)**.** For each detector, the share of the 300 per-split failure trajectories that fire *early*, *on-time* (within $\pm S$ of $t^\star$; Eq. (6)), *late*, or not at all (*no-fire*), on (a) seen and (b) unseen tasks. **FailMoment** (bottom) lands on-time on 71%/58% of seen/unseen trajectories, whereas I-FailSense and RoboFAC fire overwhelmingly early and GPT-4o/Qwen2.5-VL-3B seldom fire at all; segment labels are percentages.

On-time uses the same 300 failure trajectories per split as the trajectory-level evaluation. The success trajectories are excluded, since on-time is undefined for them.

Evaluated exactly as it runs in deployment, the detector stops at its first fire and scores no later frame. Figure 6 shows that I-FailSense and RoboFAC fire far too early (a median of 33 to 67 frames ahead of $t^\star$), while Qwen2.5-VL-3B never reaches τ (no-fire 100%) and GPT-4o stays silent on most trajectories (no-fire 75% seen, 87% unseen) and fires late on the rest. Either way the baselines rarely land in the *on-time zone*, whereas **FailMoment** fires within one or two frames of the failure and lands on-time on 71% of seen and 58% of unseen trajectories. Appendix E reports threshold-dependent precision, recall, and accuracy across τ.

D. Ablations

We ablate four design choices: backbone scale (Qwen2.5-VL-3B vs. Qwen3-VL-4B [3]), window length ($W \in \{32, 48\}$), adapter placement (LM LoRA, LM+ViT LoRA, and grounding/head variants), and the asymmetric-loss weight $\beta \in \{1.0, 2.0, 3.0\}$ in Eq. 5. The best configuration is Qwen2.5-VL-3B with W=32, LoRA on both the language model and vision encoder, and β=3.0, which is the configuration used by **FailMoment**. At the 20Hz capture rate, W=32 covers 1.6 s, providing a short recent context rather than the full trajectory. Increasing the window to W=48 (2.4 s) does not help, with unseen frame-level AUPRC decreasing from 0.871 to 0.802.

Adding vision-encoder LoRA to language-model LoRA gives the largest gain in unseen frame-level AUPRC, from 0.672 to 0.871. In contrast, adding the extra cross-modal grounding and attention-pool heads does not improve over this setting, yielding unseen frame-level AUPRC between 0.674 and 0.782. Increasing backbone scale also does not help, with Qwen3-VL-4B reaching 0.707 unseen frame-level AUPRC. For the asymmetric loss weight, β=1.0 and β=2.0 reach 0.791 and 0.778 unseen frame-level AUPRC, respectively, both below the β=3.0 setting used by **FailMoment**. Full per-cell results are reported in Appendix F.

E. Inference latency

A real-time failure detector must keep pace with the policy during execution. If the detector is slower than the control loop, the policy may pass the failure moment before the alarm can fire. **FailMoment** processes a fixed-length window of W frames at each step, so its per-step cost is constant with respect to trajectory length, $O(W)$ rather than $O(T)$. This differs from whole-trajectory detectors, whose grid or video input grows with the completed rollout and whose predictions are only available after execution [7, 21]. Since these baselines are designed for post hoc trajectory-level prediction, we report per-step latency only for **FailMoment**.

We measure inference latency on a single NVIDIA A100 80GB at batch size 1, using bfloat16(bf16) and FlashAttention-2. This matches the online setting in which one window is scored at each detector call, without additional inference-specific optimization. Latency is measured over 20 forward passes across five sample windows with CUDA synchronization, and we report both the mean and the 95th-percentile latency. At capture rate f and stride S, the detector has a budget of S/f seconds between calls. Both RLBench and LIBERO are collected at f=20Hz, so our stride S=10 gives a 500 ms budget. We benchmark the two simulators separately because their native image resolutions differ, 256×256 for RLBench and 128×128 for LIBERO, which changes the number of vision tokens. Figure 7 reports the results.

FailMoment remains within the real-time budget on both datasets. On RLBench, it runs in 187 ms per detector call, corresponding to 5.3Hz and 37% of the 500 ms budget. On

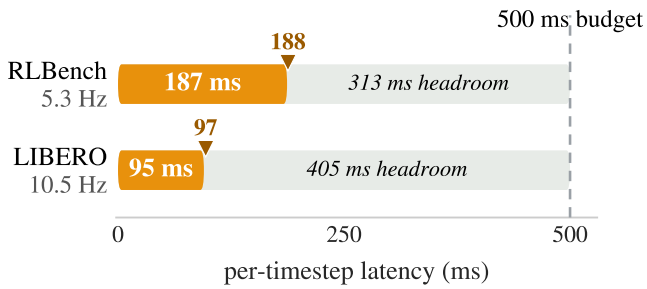


Fig. 7: **Real-time feasibility.** Per-timestep latency of **FailMoment** (A100 80GB, batch 1, bf16 + FlashAttention-2) against the 500 ms stride budget (S=10, f=20Hz). Bars are mean latency; the ▼ marks the 95th-percentile tail.

LIBERO, it runs in 95 ms, corresponding to 10.5Hz and 19% of the budget. The tail latency is also stable: the 95th percentile is 188 ms on RLBench and 97 ms on LIBERO, staying within 2 ms of the mean in both cases. Thus, even the slow tail clears the required 2Hz detector rate. The remaining headroom is at least 313 ms per call, leaving time for the policy and motion controller within the same execution loop.

F. Sim-to-real transfer on RT-1

We next evaluate whether **FailMoment** transfers to a real robot. RT-1 [4] is a large-scale real-robot dataset of everyday kitchen and tabletop tasks, recorded on physical Everyday Robots from a single third-person camera under templated language instructions. Its logged gripper signal supplies the failure frame $t^\star$, and its variation in object, color, and position supports all three of our semantic failure modes at once. We apply the relabeling of Section III to RT-1 unchanged, producing the real set $\mathcal{S}_{\text{real}}$.

Real semantic failures. Each mode is built from a single RT-1 task. *Wrong position* uses *"open the {top, middle, bottom} drawer"*; the other two levels are both valid wrong targets, meaning each trajectory yields two failures, grounded automatically since the three drawers always co-occur. *Wrong object* swaps the grasped object X in *"move X near Y"* with a distractor Z, a third object present in the scene but left untouched. Since RT-1 does not annotate this distractor, we first compile an object vocabulary from all RT-1 instructions. Then, Qwen2.5-VL-7B [2] selects the object present in the scene but absent from the instruction from this vocabulary, and the result is verified by human review. *Wrong color* requires the same object to appear in multiple colors. For episodes containing chip bags of different colors (blue, brown, and green), the instruction *"move X near Y"* is rewritten as *"grasp the {color} chip bag"*. Such scenes are rare, so they are augmented by cropping the frame at several horizontal offsets and flipping it left-to-right. Appendix C lists all RT-1 swap targets per mode.

Across the three modes, $\mathcal{S}_{\text{real}}$ comprises 1,098 success and 1,627 failure trajectories, with 397 success and 794 failures for wrong position, 397 and 397 for wrong object, and 304 and 436 for wrong color. The wrong-color trajectories are augmented from 20 source scenes.

TABLE II: **Sim-to-real transfer on RT-1.** Per-mode AUROC/AUPRC and on-time rate (τ=0.7) on the held-out real test split: **FailMoment** *frozen* (zero-shot) vs. *fine-tuned* (**FailMoment-RT-1**). On-time rates at other τ are reported in Appendix G (Table VII).

	Mode	Trajectory-level AUROC	Trajectory-level AUPRC	Frame-level AUROC	Frame-level AUPRC	On-time rate
FailMoment (Frozen)	color	0.922	0.809	0.860	0.528	10%
	object	0.733	0.584	0.686	0.358	3%
	position	0.695	0.644	0.542	0.385	3%
	mean	0.783	0.679	0.696	0.424	5%
FailMoment -RT-1	color	0.997	0.991	0.959	0.803	96%
	object	0.986	0.975	0.948	0.777	92%
	position	0.991	0.987	0.886	0.744	72%
	mean	**0.991**	**0.984**	**0.931**	**0.775**	**83%**

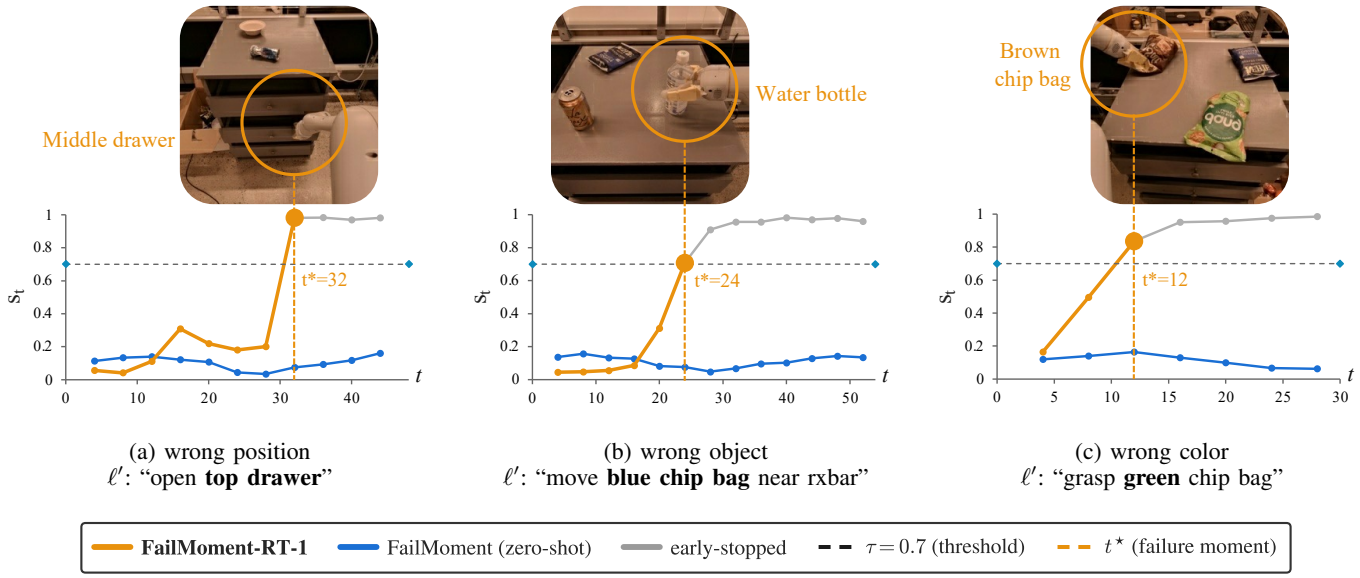
Fine-tuning and evaluation. To transfer to the real robot, we fine-tune **FailMoment** on $\mathcal{S}_{\text{real}}$, producing **FailMoment-RT-1**. We initialize it by loading only the LoRA adapter and head weights and resetting the optimizer and learning-rate schedule, keeping the same vision-encoder LoRA recipe (Section IV). The sim-learned representation thus seeds training, and AdamW then re-adapts it to the real data afresh. RT-1 clips are much shorter than our simulated trajectories (median frames $\approx$31– 33 against 164, Section III-D), so we scale the window to $W = 16$ at stride 4. Because the frozen detector under-fires on real data, we lower β to 2.0, below simulation's 3.0. Within each mode we split trajectories 80/20 into train and test, holding out 10% of train for validation. We report the trajectory-level, frame-level, and on-time metrics of Sections V-A–V-C per mode (Table II).

Results. Table II compares the frozen sim-trained detector and the RT-1 fine-tuned detector on the held-out real test split. Without fine-tuning, the frozen detector achieves a mean trajectory-level AUROC of 0.783, but its mean frame-level AUROC is 0.696 and its on-time rate is only 5%. After fine-tuning on $\mathcal{S}_{\text{real}}$, **FailMoment-RT-1** raises the mean trajectory-level AUROC to 0.991, the mean frame-level AUROC to 0.931, and the on-time rate to 83%.

Performance improves across all three semantic modes, but the gains differ by mode. The largest frame-level AUROC gain appears in wrong position, increasing from 0.542 to 0.886, followed by wrong object from 0.686 to 0.948 and wrong color from 0.860 to 0.959. For on-time detection, wrong object rises from 3% to 92%, wrong color from 10% to 96%, and wrong position from 3% to 72%.

VI. Discussion

In this work, we reframe semantic failure detection from a binary question of *"did it fail?"* to a temporal localization of *"when did it fail?"*. The resulting detector retains trajectory-level performance compared to baselines [7, 21], and addition-



(a) wrong position
ℓ': "open **top drawer**"

(b) wrong object
ℓ': "move **blue chip bag** near rxbar"

(c) wrong color
ℓ': "grasp **green** chip bag"

Fig. 8: **Detecting real RT-1 failures, frame by frame.** Examples on relabeled RT-1 trajectories for (a) wrong-position, (b) wrong-object, and (c) wrong-color failures. The boxed frame marks $t^\star$, when the robot commits to a target inconsistent with the instruction ℓ'. **FailMoment-RT-1** fires at $t^\star$ (τ=0.7), while the frozen sim-trained detector stays below threshold.

ally provides both frame-level and real-time failure prediction, as well as transfer to real-robot deployments. This suggests that *focusing on the moment* preserves trajectory-level performance and enables finer-grained capabilities.

When evaluated on frame-level tasks, the performance gap between our method and whole-trajectory baselines widens significantly. **FailMoment** successfully isolates the failure frame across all three modes on both seen and unseen tasks, whereas baselines perform near random chance, even after fine-tuning on our data. We attribute this success to the paradigm of *focusing on the moment*, where each frame is evaluated using only recent context, enabling effective tracking of failures. Both our own Qwen2.5-VL-3B backbone and the state-of-the-art GPT-4o also stay near chance zero-shot, confirming that frame-level capability stems from our window-based training rather than the backbone or its scale.

When run online under real-world deployment constraints, **FailMoment** detects failures near the moment of occurrence. In contrast, whole-trajectory baselines either trigger prematurely or fail to detect failures entirely. Notably, because each step processes a fixed-length window, the per-step cost remains constant with respect to trajectory length. This allows the detector to keep pace with the policy during execution.

Finally, the transfer to a real robot demonstrates promising sim-to-real transfer capabilities. **FailMoment**, trained only on $\mathcal{S}$, still performs above chance in trajectory-level detection on RT-1, indicating partial sim-to-real transfer for semantic failure detection. However, its weak frame-level and on-time performance shows that identifying the exact failure moment is more sensitive to the visual and interaction shift between simulation and real robot videos. After fine-tuning on $\mathcal{S}_{\text{real}}$, the sim-to-real gap largely disappears. The mode-wise gains further suggest that color grounding transfers most directly, object grounding transfers partially, and spatial or interaction grounding, as in wrong-position failures, requires the most real-domain adaptation.

The key distinction between the simulated and real-world settings lies in data preparation. While the simulator provides a comprehensive scene inventory (object identities, colors, and poses) that automates counterfactual target selection, the RT-1 dataset logs the gripper signals required for $t^\star$ but lacks this explicit inventory. Consequently, we extracted alternative targets using a VLM and manually verified them. Fully automating this pipeline is a promising direction for scaling **SemGen** and improving its real-world applicability.

VII. Conclusion

In this work, we presented **SemGen** and **FailMoment**, shifting the paradigm of semantic failure detection from a binary judgment of *whether* a rollout failed to a temporal localization of *when* it failed. To construct the dataset, **SemGen** employs counterfactual instruction relabeling, transforming successful trajectories into three failure modes. Subsequently, **FailMoment** evaluates each frame within a sliding window, enabling a single unified model to produce trajectory-level, frame-level, and real-time predictions. It remains competitive with trajectory-level baselines, yet achieves superior performance in frame-level and real-time settings. To our best knowledge, **FailMoment** is the first method to pinpoint the exact failure moment near its time of occurrence within the inference latency budget. Furthermore, our method demonstrates robust sim-to-real transfer across all three detection modes. Future work could leverage the per-frame signal for real-time recovery at the exact moment of failure.

Acknowledgments

This work was partly supported by grants from the IITP (RS-2021-II211343-GSAI/10%, RS-2022-II220951-LBA/15%, RS-2022-II220953-PICA/15%), the NRF (RS-2024-00353991-SPARC/15%, RS-2023-00274280-HEI/15%), the KEIT (RS-2025-25453780/15%), and the KIAT (RS-2025-25460896/15%), funded by the Korean government.

References

[1] Christopher Agia et al. Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress. In *Conference on Robot Learning (CoRL)*, 2024. arXiv:2410.04640.

[2] Shuai Bai et al. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025.

[3] Shuai Bai et al. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025.

[4] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Karol Gopalakrishnan, Karol Hausman, Alex Herzog, et al. RT-1: Robotics transformer for real-world control at scale. In *Robotics: Science and Systems (RSS)*, 2023.

[5] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2307.08691.

[6] Jiafei Duan et al. AHA: A vision-language-model for detecting and reasoning over failures in robotic manipulation. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2410.00371.

[7] Clémence Grislain, Hamed Rahimi, Olivier Sigaud, and Mohamed Chetouani. I-FailSense: Towards general robotic failure detection with vision-language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026. arXiv:2509.16072.

[8] Qiao Gu, Yuanliang Ju, Shengxiang Sun, Igor Gilitschenski, Haruki Nishimura, Masha Itkina, and Florian Shkurti. SAFE: Multitask failure detection for vision-language-action models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2506.09937.

[9] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. arXiv:2106.09685.

[10] Aaron Hurst et al. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[11] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. RLBench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

[12] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Conference on Robot Learning (CoRL)*, 2024.

[13] Tony Lee et al. RoboReward: General-purpose vision-language reward models for robotics. *arXiv preprint arXiv:2601.00675*, 2026.

[14] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2023.

[15] Zeyi Liu, Arpit Bahety, and Shuran Song. REFLECT: Summarizing robot experiences for failure explanation and correction. In *Conference on Robot Learning (CoRL)*, 2023. arXiv:2306.15724.

[16] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. arXiv:1711.05101.

[17] Fan Ma, Linchao Zhu, Yi Yang, Shengxin Zha, Gourab Kundu, Matt Feiszli, and Zheng Shou. SF-Net: Single-frame supervision for temporal action localization. In *Computer Vision – ECCV 2020*, pages 420–437. Springer, 2020. doi: 10.1007/978-3-030-58548-8_25.

[18] Ralf Römer et al. Failure prediction at runtime for generative robot policies. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2510.09459.

[19] Andreas Steiner, André Susano Pinto, Michael Tschannen, Daniel Keysers, Xiao Wang, Yonatan Bitton, et al. PaliGemma 2: A family of versatile VLMs for transfer. *arXiv preprint arXiv:2412.03555*, 2024.

[20] Chen Xu, Tony Khuong Nguyen, Emma Dixon, Christopher Rodriguez, Patrick Miller, Robert Lee, Paarth Shah, Rareş Ambrus, Haruki Nishimura, and Masha Itkina. Can we detect failures without failure data? uncertainty-aware runtime failure detection for imitation learning policies. In *Robotics: Science and Systems (RSS)*, 2025. arXiv:2503.08558.

[21] Zewei Ye et al. RoboFAC: A comprehensive framework for robotic failure analysis and correction. *arXiv preprint arXiv:2505.12224*, 2025.

APPENDIX

A. Dataset tasks and splits

The dataset spans 80 manipulation tasks drawn from RLBench and LIBERO, partitioned into *seen* tasks for training and *unseen* tasks held out entirely. The seen-task evaluation draws held-out trajectories from the seen tasks (familiar task and scene, new rollout). The unseen-task evaluation draws every trajectory from the held-out tasks, which contribute no frame to training. The held-out set spans all three failure modes on both simulators. Table III lists the RLBench and LIBERO tasks, each by seen/unseen split.

B. Trajectory samples

Figure 9 shows one relabeled trajectory per source and failure mode, drawn from $\mathcal{S}$ (RLBench, LIBERO) and $\mathcal{S}_{\text{real}}$ (RT-1). Each row is a single trajectory sampled at six evenly spaced timesteps, left to right, with the timestep t printed in the lower-left of every frame and the failure frame marked $t^\star$ (cf. Section III). The samples span all three modes (wrong color, object, position) across both the simulated and the real source, illustrating that the same **SemGen** relabeling and the same $t^\star$ read-out from the gripper signal apply unchanged from simulation to the real robot.

C. Swap vocabularies

Every counterfactual replaces a referring expression with another entity present in the same scene, so the pool of swap targets is determined by the scene contents rather than authored by hand. Table IV summarizes the complete vocabulary of distinct swapped-in targets for each source and intervention mode, reported in the canonical forms used by the relabeled instructions. The available modes vary by source according to what each environment exposes. RLBench provides color and position interventions together with a small object vocabulary, LIBERO provides object interventions, and RT-1 provides all three intervention modes. Notably, the position vocabulary is identical in simulation and on the real robot (*top drawer*, *middle drawer*, and *bottom drawer*), enabling direct transfer of position interventions across domains.

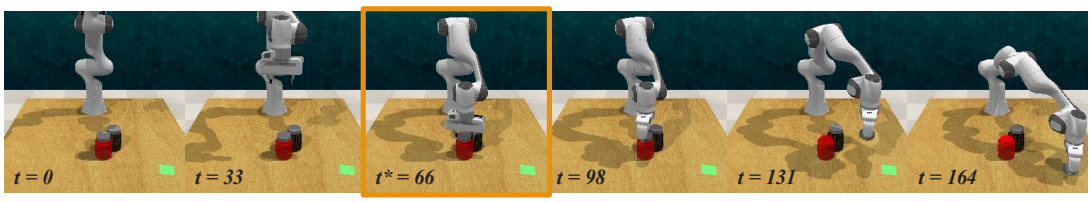
The object names listed in Table IV are canonicalized to improve the readability and consistency of the relabeled instructions. In LIBERO, instance identifiers, SKU numbers, and brand prefixes are removed; for example, `chefmate_8_frypan` is relabeled as "frypan." Some names are also rewritten to better reflect natural language usage. For instance, the two-tone `white_yellow_mug` becomes "yellow mug," since "white yellow mug" may be interpreted as referring to two separate mugs. In RT-1, whose frames are recorded at a resolution of 256×256, visually indistinguishable object variants are merged into a single label. For example, green rice and green jalapeño chip bags are both relabeled as "green chip bag," while `rxbar_chocolate` and `rxbar_blueberry` are both shortened to "rxbar."

TABLE III: **Tasks of $\mathcal{S}$ by simulator and split.** Seen tasks are for training, unseen tasks are fully held out. RLBench supplies the wrong-color and wrong-position modes; LIBERO tasks are summarized by task family, grouped by scene (Kitchen, Living Room, Study), and the source of wrong-object failures.

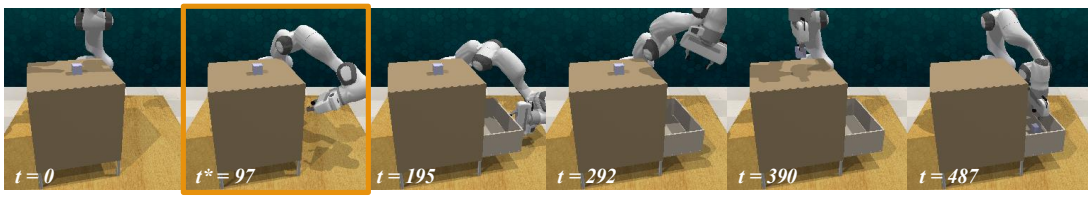
RLBench	
Seen	close drawer; insert onto square peg; light bulb in; light bulb out; meat off grill; open jar; pick up cup; pour from cup to cup; put item in drawer; stack cups; take item out of drawer.
Unseen	close jar; meat on grill; open drawer; stack blocks.
LIBERO	
Seen	**Kitchen:** place or stack black/white bowls on the cabinet, plate, or drawer; put the moka pot on the stove; put the wine bottle in the drawer or on the rack; put the ketchup in the drawer; put the yellow-and-white mug in the microwave. **Living Room:** pick up a pantry item (alphabet soup, butter, milk, orange juice, tomato sauce, ketchup, cream cheese, chocolate pudding, salad dressing) and put it in the basket or tray, singly or in pairs; stack left/right bowls into the tray; place red/white/yellow mugs on the left/right plate; position the chocolate pudding left/right of the plate. **Study:** place a book in the front/back/left/right compartment of the caddy or on the cabinet shelf; place red/white/yellow-and-white mugs beside the caddy.
Unseen	**Kitchen:** put the black bowl on top of the cabinet; turn on the stove and put the moka pot on it. **Living Room:** put both the alphabet soup and the tomato sauce in the basket. **Study:** place a book in the front compartment of the caddy.

TABLE IV: **Swap vocabularies by source and mode.** Distinct swapped-in targets per cell (count in parentheses), in the canonical forms used in the relabeled instructions.

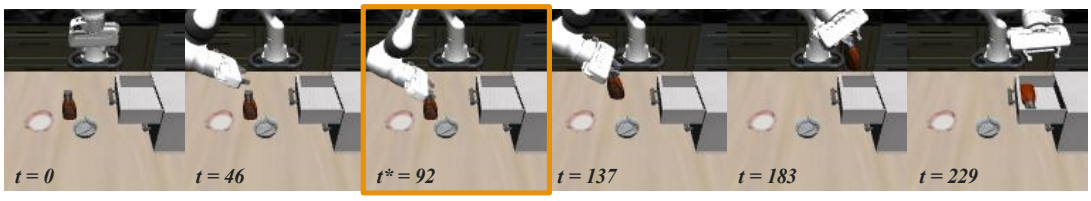
Source	Mode	Swapped-in targets
RLBench	color (20)	azure, black, blue, cyan, gray, green, lime, magenta, maroon, navy, olive, orange, purple, red, rose, silver, teal, violet, white, yellow
	object (2)	chicken, steak
	position (3)	bottom drawer, middle drawer, top drawer
LIBERO	object (18)	akita black bowl, alphabet soup, black book, butter, chocolate pudding, cream cheese, frypan, ketchup, milk, new salad dressing, orange juice, plate, porcelain mug, red coffee mug, tomato sauce, wine bottle, yellow book, yellow mug
RT-1	color (3)	blue chip bag, brown chip bag, green chip bag
	object (15)	7up can, apple, blue chip bag, blue plastic bottle, brown chip bag, coke can, green can, green chip bag, orange, orange can, pepsi can, redbull can, rxbar, sponge, water bottle
	position (3)	bottom drawer, middle drawer, top drawer



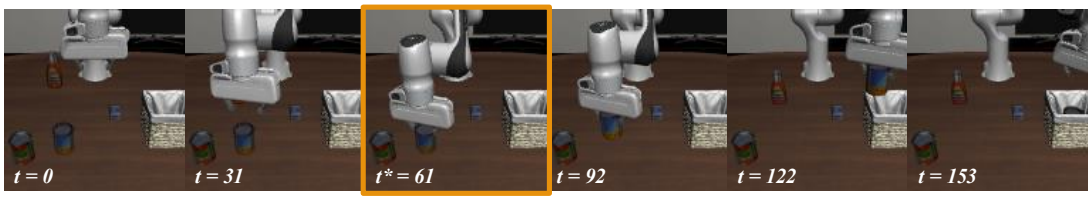
(a) **RLBench, wrong color.** *"open the* ***red*** *jar"* → *"open the* ***black*** *jar".*



(b) **RLBench, wrong position.** *"put the item in the* ***bottom*** *drawer"* → *"put the item in the* ***middle*** *drawer".*

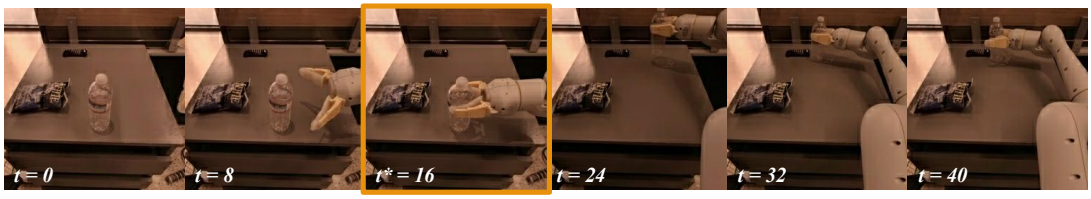


(c) **LIBERO, wrong object.** *"put the* ***ketchup*** *in the top drawer"* → *"put the* ***black bowl*** *in the top drawer".*

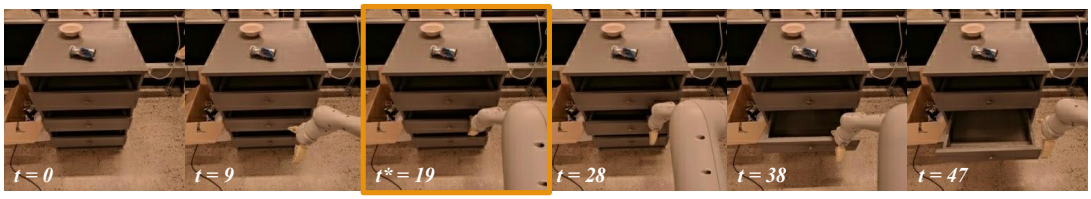


(d) **LIBERO, wrong object.** *"pick up the* ***alphabet soup*** *and put it in the basket"* → *"pick up the* ***cream cheese*** *and put it in the basket".*

(e) **RT-1, wrong color.** *"grasp the* ***brown*** *chip bag"* → *"grasp the* ***green*** *chip bag".*



(f) **RT-1, wrong object.** *"move the* ***water bottle*** *near rxbar"* → *"move the* ***blue chip bag*** *near rxbar".*



(g) **RT-1, wrong position.** *"open the* ***middle*** *drawer"* → *"open the* ***top*** *drawer".*

Fig. 9: **Trajectory samples from $\mathcal{S}$ and $\mathcal{S}_{\text{real}}$.** One relabeled trajectory per source and failure mode, sampled at six evenly spaced timesteps (left to right); the timestep t is printed in the lower-left of each frame and the failure frame is marked $t^\star$. Rows (a)–(d) are simulated ($\mathcal{S}$: RLBench, LIBERO) and rows (e)–(g) are real ($\mathcal{S}_{\text{real}}$: RT-1), together covering all three semantic modes (wrong color, object, position). In each subcaption the original target (ℓ) is shown in **blue** and the swapped-in distractor (ℓ') in **orange**, as in Figure 3.

D. Detector prompt

FailMoment uses one fixed prompt for both training and inference, applied through the Qwen2.5-VL chat template with `add_generation_prompt` disabled. The most recent W frames enter as the video content and the instruction ℓ fills the user text. No tokens are generated — the linear head reads the hidden state of the final prompt token (Section IV). The system and user messages are:

```
[system]
You are an online failure detector for
robotic manipulation. Observe the video
so far and estimate whether a failure has
occurred in the latest moment.

[user]
{W video frames}  Task: "{instruction}"
```

E. Threshold-based metrics across τ

Table V reports the threshold-dependent metrics, precision (P), recall (R), accuracy (Acc), and, for the trajectory level, the on-time rate (OT), swept over $\tau \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$, mode-pooled, for the trajectory- and frame-level evaluations of Section IV. How sharply these metrics react to τ is governed by the shape of each method's score distribution, and the methods fall into three regimes. **FailMoment**'s scores are sharply bimodal (cf. Figure 6), making it essentially invariant to τ. Frame-level metrics shift by $\leq$ 0.007 and the on-time rate by $\leq$ 3 points across the entire range, and any threshold in the wide mid-band yields the same decisions. I-FailSense and RoboFAC, whose scores are diffuse, swing strongly instead (e.g. RoboFAC (fine-tuned) frame-level recall on unseen tasks falls from 0.46 to 0.09 as τ goes $0.5 \rightarrow 0.9$). The raw Qwen2.5-VL-3B backbone collapses the other way. Its scores are compressed below 0.6, and for $\tau \geq 0.6$ it predicts all-negative on every pool, with recall zero and precision undefined (– -). GPT-4o, also zero-shot, sits between these regimes. Its scores are likewise compressed, so recall stays low and fades with τ (seen frame-level $0.066 \rightarrow 0.025$), yet a thin firing tail keeps it from collapsing fully to all-negative. The flatness throughout means the main text's choice of $\tau = 0.7$ is not a tuned hyperparameter but simply a mildly conservative point within that flat region.

F. Ablations

The four factors of Section V-D are ablated around our configuration (Qwen2.5-VL-3B, W=32, LoRA on both the language model and the vision encoder), each variant taken at its best validation checkpoint and scored by overall (mode-pooled) frame-level AUROC and AUPRC on both splits.

Backbone, window, and adapter placement. The first three factors are reported in Table VI. Vision-encoder LoRA is the decisive ingredient. Adding it lifts unseen AUPRC from 0.672 to 0.871. In contrast, a longer window, a larger and more recent backbone, and the extra cross-modal grounding and attention-pool modules each fail to improve localization. **FailMoment** adds only LoRA adapters and a linear head ($\approx$ 23M trainable parameters, 0.6% of the backbone), yet matches or beats every larger or heavier setting. The smallest, cheapest configuration is also the strongest.

This points to a diagnostic beyond hyperparameter tuning. The bottleneck of semantic failure detection sits in the vision encoder, not in the language model and not in backbone scale. The newer, larger Qwen3-VL-4B (with a redesigned vision encoder) does not improve localization, and adding cross-modal modules or widening the window above the language model does not either. Adapting the vision encoder of the existing 3B backbone at r=32, by contrast, closes most of the seen–unseen gap (AUPRC $0.672 \rightarrow 0.871$). Web-pretrained vision encoders do not, out of the box, separate the distractors that semantic failures hinge on (the wrong cup, the wrong drawer, the wrong colored object) with enough fidelity to support frame-level localization, and LM-side adaptation do not supply what the vision side does not encode. Since VLAs themselves sit on the same kind of vision encoder, vision-side adaptation is a natural leverage point for grounding-sensitive downstream tasks in general.

Asymmetric loss weight. The negative weight β of the asymmetric BCE (Eq. 5) is swept next, holding the main configuration fixed (Qwen2.5-VL-3B, W=32, LM + ViT) over $\beta \in \{1.0, 2.0, 3.0\}$, where β=1.0 recovers the symmetric loss and β=3.0 is **FailMoment**. The loss-weight block of Table VI reports overall frame-level AUROC and AUPRC at β=1.0 and 2.0, with β=3.0 the anchor **FailMoment** row. **FailMoment**'s β=3.0 is best on every cell, most clearly on unseen AUPRC (0.871, against 0.791 at β=1.0 and 0.778 at β=2.0). The intermediate weight does not improve on the symmetric loss, and the gain therefore comes from the stronger asymmetry rather than a monotone trend in β, consistent with the deployment rationale of Eq. 5, where a confident-firing bias trades a little recall for far fewer false alarms.

G. Real-robot threshold robustness

The same threshold analysis on the real robot confirms that the τ-invariance carries over from simulation. Table VII sweeps the RT-1 on-time rate (Section V-F, stride S=4) over $\tau \in \{0.5, \ldots, 0.9\}$, per mode and as the mode mean, for **FailMoment** frozen and fine-tuned (**FailMoment-RT-1**). **FailMoment-RT-1**'s mean on-time rate holds at 83% across the entire range, and every per-mode entry varies by at most one point (the τ-range spread column); the sharply bimodal score regime that makes the simulated detector τ-invariant thus transfers intact to real data. The frozen detector stays near zero at every threshold, never reaching the on-time band. The $\tau = 0.7$ reported in the main-text Table II is therefore, as in simulation, a mildly conservative point inside a flat band rather than a tuned value.

TABLE V: **Threshold-dependent metrics vs. τ.** Mode-pooled precision (P), recall (R), accuracy (Acc), and on-time rate (OT, trajectory-level only), for the trajectory- and frame-level evaluations, on the seen-task and unseen-task splits; companion to Table I. Our detector is essentially flat across τ while the baselines either swing strongly or collapse to all-negative.

		Trajectory-level								**Frame-level**					
		seen task				unseen task				seen task			unseen task		
Method	τ	P	R	Acc	OT	P	R	Acc	OT	P	R	Acc	P	R	Acc
Qwen2.5-VL-3B [2]	0.5	0.620	0.387	0.575	0.050	0.493	0.227	0.497	0.020	0.208	0.053	0.735	0.178	0.033	0.818
	0.6	–	0.000	0.500	0.000	–	0.000	0.500	0.000	–	0.000	0.770	–	0.000	0.838
	0.7	–	0.000	0.500	0.000	–	0.000	0.500	0.000	–	0.000	0.770	–	0.000	0.838
	0.8	–	0.000	0.500	0.000	–	0.000	0.500	0.000	–	0.000	0.770	–	0.000	0.838
	0.9	–	0.000	0.500	0.000	–	0.000	0.500	0.000	–	0.000	0.770	–	0.000	0.838
GPT-4o [10]	0.5	0.515	0.353	0.510	0.087	0.684	0.173	0.547	0.047	0.325	0.066	0.753	0.536	0.031	0.838
	0.6	0.516	0.313	0.510	0.073	0.687	0.153	0.542	0.043	0.344	0.056	0.758	0.522	0.025	0.838
	0.7	0.500	0.253	0.500	0.067	0.684	0.130	0.535	0.033	0.374	0.049	0.762	0.500	0.016	0.838
	0.8	0.504	0.207	0.502	0.057	0.659	0.097	0.523	0.020	0.403	0.041	0.765	0.462	0.012	0.837
	0.9	0.454	0.130	0.487	0.037	0.667	0.067	0.517	0.010	0.358	0.025	0.765	0.500	0.008	0.838
I-FailSense [7]	0.5	0.848	0.763	0.813	0.010	0.616	0.833	0.657	0.003	0.270	0.651	0.514	0.180	0.922	0.305
	0.6	0.861	0.743	0.812	0.010	0.629	0.780	0.660	0.003	0.274	0.640	0.526	0.183	0.878	0.344
	0.7	0.869	0.707	0.800	0.010	0.719	0.580	0.677	0.000	0.281	0.606	0.552	0.195	0.755	0.453
	0.8	0.890	0.673	0.795	0.010	0.736	0.520	0.667	0.000	0.289	0.595	0.569	0.193	0.680	0.487
	0.9	0.909	0.630	0.783	0.007	0.770	0.470	0.665	0.000	0.296	0.579	0.585	0.194	0.621	0.519
RoboFAC [21]	0.5	0.686	0.800	0.717	0.057	0.567	0.710	0.583	0.090	0.289	0.517	0.595	0.205	0.460	0.622
	0.6	0.683	0.690	0.685	0.043	0.585	0.610	0.588	0.093	0.308	0.491	0.629	0.186	0.307	0.670
	0.7	0.682	0.637	0.670	0.033	0.612	0.463	0.585	0.077	0.339	0.461	0.669	0.174	0.194	0.720
	0.8	0.687	0.520	0.642	0.047	0.686	0.320	0.587	0.053	0.381	0.405	0.712	0.200	0.148	0.765
	0.9	0.696	0.367	0.603	0.060	0.871	0.247	0.605	0.030	0.416	0.297	0.742	0.201	0.089	0.795
FailMoment	0.5	0.896	0.890	0.893	0.707	0.638	0.870	0.688	0.570	0.792	0.929	0.928	0.732	0.885	0.929
	0.6	0.896	0.890	0.893	0.707	0.640	0.870	0.690	0.577	0.792	0.928	0.927	0.732	0.885	0.929
	0.7	0.896	0.890	0.893	0.707	0.640	0.870	0.690	0.577	0.793	0.928	0.928	0.733	0.885	0.929
	0.8	0.895	0.880	0.888	0.693	0.637	0.860	0.685	0.573	0.793	0.926	0.927	0.735	0.882	0.929
	0.9	0.897	0.867	0.883	0.683	0.640	0.843	0.685	0.560	0.795	0.926	0.928	0.739	0.880	0.930

TABLE VI: **Frame-level ablation.** Each block varies a single axis around the **FailMoment** setting (bold anchor row). Adapter codes: *LM* = LoRA on the language model; *ViT* = on the vision encoder; *G/P* = cross-modal grounding + attention-pool head; *full* = ViT + G/P. Best per column in bold.

Variant		AUROC		AUPRC	
Factor	Setting	seen	unseen	seen	unseen
FailMoment	3B, $W = 32$, LM + ViT, $\beta = 3.0$	**0.971**	**0.969**	**0.864**	**0.871**
Adapter	LM only	0.946	0.909	0.800	0.672
	LM + G/P	0.944	0.909	0.779	0.674
	full (ViT + G/P)	0.963	0.950	0.844	0.782
Window W	W=48, LM + ViT	0.963	0.944	0.861	0.802
	W=48, full	0.956	0.958	0.821	0.784
Backbone	Qwen3-VL-4B, LM + ViT	0.956	0.939	0.807	0.707
	Qwen3-VL-4B, full	0.943	0.937	0.798	0.689
Loss β	β=1.0 (symmetric)	0.966	0.950	0.847	0.791
	β=2.0	0.953	0.944	0.830	0.778

TABLE VII: **RT-1 on-time rate vs. τ.** Per-mode and mean on-time rate (%) on the held-out real test split (Section V-F, S=4), swept over $\tau \in \{0.5, \ldots, 0.9\}$, for **FailMoment** *frozen* (zero-shot) and *fine-tuned* (**FailMoment-RT-1**); companion to Table II (τ=0.7 column). The rate is essentially flat across τ: **FailMoment-RT-1** holds its 83% mean throughout while the frozen detector stays near zero.

	Mode	Threshold τ					τ-range spread
		0.5	0.6	0.7	0.8	0.9	
FailMoment (Frozen)	color	10	10	10	7	6	4
	object	4	4	3	3	1	3
	position	3	3	3	3	3	0
	mean	5	5	4	4	3	2
FailMoment -RT-1	color	96	96	96	96	95	1
	object	92	92	92	92	92	0
	position	72	72	72	72	72	0
	mean	**83**	**83**	**83**	**83**	**83**	0