

RoboFailRing: Retrieval-Augmented and Language Grounding Failure Detection for VLM-enabled Robotic Manipulation

Chenduo Ying¹, Linkang Du², Yuanchao Shu^{1*}, Peng Cheng¹

¹Zhejiang University ²Xi'an Jiaotong University

Abstract

Reliable failure detection and causal reasoning are critical in robotic manipulation, as their absence risks robot damage and endangers human safety. Although recent Vision-Language Models (VLMs) are employed to attempt failure detection and causality reasoning, they typically make retrospective assessment only after task completion, and their reasoning accuracy is often limited. To address these issues, we introduce **RoboFailRing**, which enables timely failure detection during task execution and enhances the reasoning accuracy of VLMs. It achieves rapid failure detection by retrieving a pre-constructed failure memory and returning a similarity-based decision. In addition, by providing grounded failure report to VLMs, it improves the accuracy of their reasoning about the failure causes and repair strategies. We evaluate RoboFailRing on two large-scale simulated datasets comprising over 6,000 failure trajectories and covering 81 distinct manipulation tasks. The results show that the average success rate of out-of-distribution failure detection reaches 80%, while the mean detection time is cut to roughly 50% of the baseline. Moreover, evaluations on real-world systems show an average 35% gain in VLM failure-reasoning accuracy. We make our code publicly available at: <https://github.com/DynamicPoet/RoboFailRing>.

1 Introduction

Accompanied by the substantial advances in foundation models (Firoozi et al., 2025; Zheng et al., 2025) in recent years, integrating Large Language Models (LLMs) and Vision-Language Models (VLMs) into embodied agents such as robots fundamentally reshapes the landscape of robotic manipulation. By harnessing the understanding and reasoning capabilities of LLMs and VLMs, robots are now able to execute open-ended instructions

*Corresponding author.

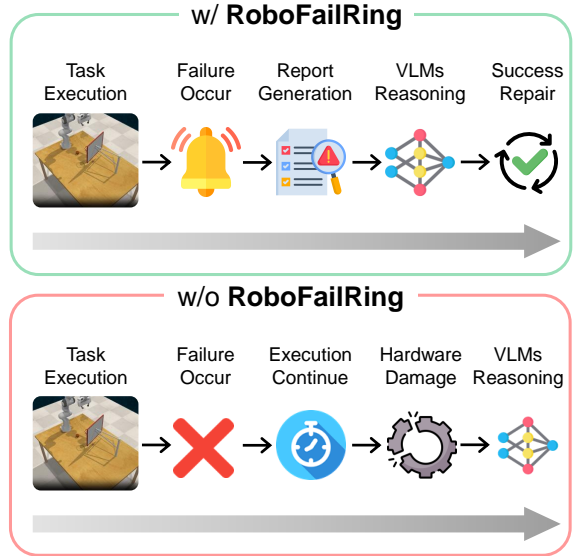


Figure 1: **VLM-based failure detection.** **RoboFailRing** enables successful repair via proactive warnings (top), avoiding the hardware damage caused by retrospective analysis (bottom).

and generate complex task policies (Octo Model Team et al., 2024; Chi et al., 2025) as well as executable code (Liang et al., 2023; Huang et al., 2023) across a wide range of tasks. However, an essential requirement for robots to successfully execute and complete tasks is their capability to monitor and detect task states. A reliable robot can not only detect failures but also reason about their causes. Currently, VLMs serve as the core responsible for detecting failures, due to their proficiency in multi-modal perception and reasoning among other components.

Despite their advantages in detection and reasoning, VLMs still face challenges when performing failure detection in practice. **The main limitation is the time of detection.** Most VLMs-based failure detection waits for task completion because failure is defined by the final outcome and VLMs require full task semantic context to assess. Therefore,

regardless of whether the task has already failed during execution, the VLM is unable to perform detection while executing. As indicated by the red box in Figure 1, this paradigm of retrospective analysis may lead to irreversible hardware damage and may even pose risks to human safety. **Moreover, VLMs exhibit inadequate reasoning capabilities for the failures in robotic manipulation.** In the absence of task-specific context, VLMs struggle to distinguish between two visually similar failure modes (e.g., "*rotation error*" and "*translation error*"), which can lead to reasoning outcomes that appear plausible but are incorrect. These challenges stimulate the research community’s attention to reliable failure detection and safety in LLM/VLM-enabled robotic manipulation (Duan et al., 2024; Ma et al., 2023; Ha et al., 2023; Ying et al., 2025). While these studies acknowledge the critical role of failure detection and safety in robotic manipulation, practical methods with specific technical details are still absent. Recent works (Kim et al., 2023; Yoo et al., 2024; Zhu et al., 2024) employ retrieval augmentation or memory construction to enhance models’ perception and planning capabilities, but they do not focus on failure detection. In practice, we should move from a retrospective analysis failure-detection paradigm to a proactive warning framework and improve VLM reasoning accuracy by supplying grounded task-specific context.

In this paper, we present **RoboFailRing**, a generic framework designed to enable rapid failure detection and generate grounded failure report for VLM-enabled robotic manipulation. Specifically, RoboFailRing achieves rapid failure detection by retrieving a pre-constructed failure memory and returning a similarity-based decision. However, building a failure memory from large-scale failure trajectories is challenging. Failure trajectories are often diverse and sparse, and the contextual states vary across different tasks. We construct failure memory by encoding failure trajectory images as concatenated embeddings of an initial task context frame and final failure state frame. In addition, we introduce a neuro-symbolic grounding mechanism that grounds concise and abstract task metadata into natural language useful for VLM reasoning. The resulting natural language forms part of the failure report. By providing grounded failure report to VLMs, it improves the accuracy of their reasoning about the causes of failures and the corresponding repair strategies.

In summary, our contributions are three-fold:

- We introduce **RoboFailRing**, a generic framework that builds a failure memory by embedding failure trajectories and associating them with task metadata, thereby enabling rapid failure detection.
- We propose a mechanism to generate and exploit failure reports, which supply grounded failure context and reference images. Leveraging this information enables the VLM to reason about failures more accurately.
- We conduct comprehensive evaluations on two large-scale simulation datasets and real-world systems. Our framework attains an average failure-detection success rate of 80% and reduces detection time by 50% relative to the baseline. In addition, it improves the average reasoning accuracy of VLM by 35%.

2 Related Work

2.1 VLMs for Robotic Manipulation

Significant progress in vision-language models has opened new possibilities for robotic manipulation (Firoozi et al., 2025). These foundation models integrate visual perception and language understanding, endowing robots with open-vocabulary understanding and cross-task generalization (Gao et al., 2024; Din et al., 2025). Recent representative systems (e.g., RT-1 (Brohan et al., 2023), PaLM-E (Driess et al., 2023), RoboPoint (Yuan et al., 2024) and Pelican-VL (Zhang et al., 2025)) integrate visual and language encoders into a unified controller. These approaches build upon the ability of VLMs to provide high-level commonsense reasoning and use visual inputs as the foundation for task-relevant representations, enabling robots to execute a wide range of tasks specified by image and text prompts. Existing works that incorporate VLMs into robotic manipulation can be broadly categorized into two lines of work. One line of work treats VLMs as frozen or lightly adapted visual reasoning engines. By converting manipulation problems into visual question answering or conditional reasoning tasks, these approaches (Huang et al., 2023; Liu et al., 2024, 2025; Muttaqien et al., 2025) exploit the pretrained semantic and spatial priors of VLMs without requiring end-to-end retraining. A second line of work fine-tunes or instruction-tunes VLMs on robotics-specific data. Here, a general VLMs is

specialized to output manipulation affordances or reasoning given imagery (Yuan et al., 2024). Similarly, specialized VLMs have been trained for tactile and physical reasoning (Yu et al., 2024). Other tuned VLMs include A3VLM (Huang et al., 2024), which learns affordances from video and gesture. The underlying idea of our framework is similar to the first line, but we use historical images and grounded languages to prompt VLMs.

2.2 Failure Detection in Robotic Manipulation

Despite their demonstrated effectiveness in executing manipulation tasks, existing VLM-enabled robotic manipulation methods exhibit notable limitations in detecting, interpreting, and recovering from failures. Most current models are optimized to predict successful actions under idealized assumptions, yet real-world robotic manipulation is inherently stochastic and error-prone. Failures such as object slippage, missed grasps, unintended collisions, or spatial misalignment frequently occur, especially in long-horizon or dynamic environments. A predominant line of work (Ma et al., 2023; Wang et al., 2024; Duan et al., 2024) leverages readily available LLMs or VLMs as components to assess whether a task has been completed, and some studies (Du et al., 2023; Ma et al., 2025) further adapt these models through finetuning or value estimation to identify unsuccessful executions. While effective in practice, such approaches usually frame the problem as a binary or numerical decision and stop short of offering textual justifications, thereby failing to explain the specific reasons behind observed failures. Latest research (Duan et al., 2025) begin to address this limitation by finetuning VLMs on failure data, enabling them to determine whether a task has failed, analyze the underlying causes, and provide semantic-level explanations. However, employing such fine-tuned VLMs for failure detection poses challenges in terms of timeliness and reasoning accuracy. Our work introduces failure detection based on retrieval augmentation and language grounding, aiming to address these two challenges.

3 Methodology

In this section, we present **RoboFailRing** in detail. The implementation of the framework consists of three key components: (1) the construction of failure memory; (2) the realization of failure detection; and (3) the generation and utilization of failure reports.

3.1 Workflow Overview

As illustrated in Figure 2, **RoboFailRing** is a framework designed to detect failed manipulations and reason potential errors through failure report. The workflow begins with the task execution, where the robot manipulates under natural language task instructions. During execution, we continuously capture multi-view observations and employ a frozen encoder such as CLIP (Radford et al., 2021) to perform dual-frame fusion embedding for each viewpoint. These real-time embeddings retrieve our pre-constructed failure memory, a database storing historical failure trajectories indexed with grounded metadata. A failure detector then evaluates the similarity between the current state and retrieved failure instances. If the similarity exceeds a certain threshold, the system anticipates an imminent error and triggers task termination, while if it does not, the system continues to monitor execution and performs retrieval on newly captured frames. In the end, the system generates a structured and retrieval-augmented failure report containing relevant image frames and textual context, prompting the VLMs to do the failure reasoning. The results obtained from the reasoning of VLMs are used to guide the user or robot in performing repairs.

In the following sections, we delve into the detailed design of each module.

3.2 Failure Memory Construction

The core of **RoboFailRing** is to construct a spatio-temporal Failure Memory (\mathcal{M}). The construction process is an offline pipeline consisting of four stages. The objective of this pipeline is to transform raw failure trajectories into a semantically grounded vector space to enable rapid retrieval.

First, we address the collection of failure trajectory data. We adopt FailGen from the recent work of (Duan et al., 2025), an extensible simulation framework designed for the programmatic generation of failure trajectories. Its failure trajectory data generation offers high flexibility and meets our requirements for large-scale generation. FailGen is a custom environment wrapper designed for RL-Bench (James et al., 2020) that modifies task trajectories by perturbing keyframes, replacing objects, and reordering keyframe sequences. This wrapper systematically generates 7 categories of failure trajectories, namely incomplete grasp (*No_Grasp*) failure, inadequate grip retention (*Slip*) failure, misaligned keyframe (*Translation*) failure, incor-

Figure 2: Overview of RoboFailRing. The yellow block represents the construction of failure memory (Section 3.2). The gray blocks denote the task workspace for robot manipulation and the failure detector (Section 3.3). The red and green blocks respectively indicate the generated failure report and the prompt template used for VLM reasoning (Section 3.4). Task instructions can be adjusted to the requirements.

rect rotation (Rotation) failure, missing rotation (No_Rotation) failure, wrong action sequence causality of a failure is sufficiently encapsulated (Wrong_Sequence) failure, and wrong target object (Wrong_Object) failure. These procedurally generated large-scale robot manipulation failure trajectory datasets, spanning multiple distinct categories, ensure diversity and complexity in failure modes, and we use them as the primary source of failure memory. LeD_{raw} denote the source dataset containing N manipulation trajectories across different task types (e.g. basketball_in_hoop, open_microwave etc.). Unlike standard datasets that prioritize successful demonstrations, FailGen provides a rich distribution of failures caused by subtle deviations or physical interactions (e.g. collision-induced slippage). We perform an initial filtering operation to construct our failure corpus $D_{fail} \subseteq D_{raw}$:

$$D_{fail} = \{E_i \mid Outcome(E_i) \in \text{Success}\}_{i=1}^N \quad (1)$$

Each trajectory E_i is a tuple $(I_i; V_i; M_i)$, containing a temporal sequence of RGB images I_i , a set of synchronized camera views $V = \{V_{front}; V_{overhead}; V_{wrist}\}$, and a structured failure metadata M_i .

Second, we perform keyframe selection and handle multi-view independence. A critical design choice in our framework is the representation of time. While video transformers can process full temporal sequences, they are computationally pro-

hibitive for real-time retrieval. We observe that the causality of a failure is sufficiently encapsulated by the contrast between the initial task content and the deviation in the end of failure. Similar empirical observations are also discussed in (Liu et al., 2023). Therefore, for each failure trajectory, we extract a set of keyframe pairs. To mitigate occlusion and maximize retrieval recall, we treat each camera view as an independent memory entry. For a specific view $v \in V_i$, we define the memory unit as $U_{i;v} = (I_{start}^{(v)}; I_{end}^{(v)})$, where $I_{start}^{(v)}$ is the frame at $t = 0$ representing the task context and $I_{end}^{(v)}$ is the frame at $t = T_{fail}$ representing the failure state. Formally, a failure state is a terminal state in which the object configuration or robot pose deviates from task goal constraints (e.g., the target object outside the designated area or an empty gripper in a pick-and-place task), as determined automatically by the simulator via pre-defined goal checking algorithms. The above multi-view independent strategy ensures that, even if a failure is visible from only one of the three cameras, it can still be effectively indexed and retrieved. For example, in the task of “basketball_in_hoop”, images captured by the front-view camera are easily completely occluded by the large backboard, thereby providing no useful information, while images captured by the overhead camera are occluded by the robotic arm. Only images from the wrist-view camera can be used to determine whether the task succeeds or fails. To

bridge the modality gap between visual observations and semantic reasoning, we employ CLIP as our visual encoder. We utilize the ViT-B/32 variant that operates at runtime (See Appendix A, denoted as \mathcal{G}), which projects images into a 512-dimensional space. We explicitly reject the strategy of averaging features, which tends to wash out fine-grained failure details. Instead, we propose a dual-frame concatenated normalization scheme. For a given pair $(I_{\text{start}}; I_{\text{end}})$, we first extract and independently normalize the feature vectors:

$$f_{\text{start}} = \frac{(I_{\text{start}})}{k(I_{\text{start}})k_2}; f_{\text{end}} = \frac{(I_{\text{end}})}{k(I_{\text{end}})k_2}; \quad (2)$$

This independent normalization preserves the magnitude of features in both states, preventing one frame from dominating the representation. The final spatio-temporal memory vector $v_{\text{mem}} \in \mathbb{R}^{1024}$ is obtained by concatenating and re-projecting onto the high-dimensional space:

$$v_{\text{mem}} = \frac{f_{\text{start}} \parallel f_{\text{end}}}{k(f_{\text{start}} \parallel f_{\text{end}})k_2}; \quad (3)$$

Here, \parallel denotes vector concatenation. This resulting vector space \mathbb{R}^{1024} effectively clusters failures that share both initial task context and failure state (See Appendix D for pseudocode of the construction algorithm).

Third, we perform language grounding. A purely vector-based memory is opaque to human users and VLMs reasoner. To bridge the gap between

high-dimensional vectors and interpretable semantics, we introduce a neuro-symbolic grounding mechanism. The failure trajectories generated by RoboFailRing FailGen are annotated with concise and abstract labels, such as "rotation_x_wp1_episode0", which denotes an x-axis rotational failure occurring at a specific waypoint (wp1) in episode 0 of the current task. While such compact and abstract labels facilitate the construction of large-scale datasets, they are not well suited for the generalization reasoning required by VLMs. A naive approach would store natural language failure descriptions directly in the metadata. However, natural language is inherently ambiguous and difficult to index precisely. In view of the aforementioned problems, we adopt a neuro-symbolic grounding approach. For each indexed vector v_k , we store a structured metadata tuple $M_k = (k; y_{\text{sym}})$, where k is the task identifier, used for task-constrained search space pruning, and y_{sym} is the raw failure symbol derived from the vectors, making the inner product a valid metric for similarity. Crucially, by including v_{start} in the

as a compressed, unambiguous ground truth label. We define a deterministic semantic mapping function $\mathcal{Y} : \mathbb{L} \rightarrow \mathbb{Y}$ that operates at runtime (See Appendix C for details). This function dynamically expands the discrete symbols y_{sym} into a coherent natural language explanation $L_{\text{desc}} : \mathbb{L}_{\text{desc}} = \mathcal{G}(y_{\text{sym}})$. For example, $\mathcal{G}(\text{rotation_x_wp1}) = \text{"The failure was caused by incorrect gripper rotation along the X-axis at the initial stage (Waypoint1)"}$. This neuro-symbolic design effectively decouples the precision of symbolic storage from the expressiveness of natural language. It allows users to refine prompt engineering for VLMs without reconstructing the underlying vector index of failure memories, thereby ensuring both the flexibility of the framework and the consistency of the data.

Finally, the collection of memory vectors v_k and their associated metadata M_k are stored efficiently. We utilize Faiss (Facebook AI Similarity Search) (Douze et al., 2026) as our vector database because of its efficiency and broad support. Since all vectors v_{mem} are L_2 -normalized, the inner product search is mathematically equivalent to cosine similarity. This ensures that the retrieval metric aligns perfectly with the underlying CLIP model. The resulting memory contains thousands of embedding data from distinct failure trajectories, covers dozens manipulation tasks, and supports millisecond-level queries.

3.3 Failure Detection Mechanism

We first configure the failure memory, after which we implement the failure detection mechanism. During execution of a new task, the RoboFailRing FailGen are annotated with concise and abstract functions as a high-frequency failure detector. The role of this detector is to determine whether the manipulation results in a failure.

Let $I_{\text{stream}} = [I_0; I_1; \dots; I_t]$ be the continuous image stream captured by the cameras. At any time step, the framework maintains a buffer containing the initial state frame $I_{\text{start}} = I_0$ and the current observation $I_{\text{curr}} = I_t$. To ensure strict alignment with the offline memory M , the online query vector q_t is constructed using the identical concatenated normalization scheme defined in Section 3.2

$$q_t = \frac{(I_{\text{start}})}{k(I_{\text{start}})k_2} \parallel \frac{(I_{\text{curr}})}{k(I_{\text{curr}})k_2}; \quad (4)$$

This formulation ensures that the query lies on the same high-dimensional space as the stored memory M , making the inner product a valid metric for similarity. Crucially, by including v_{start} in the

query, we condition the retrieval on the initial environmental affordance, thereby filtering out irrelevant failure modes that may look visually similar but are causally impossible given the starting state.

Direct searching of the entire memory can cause cross-task interference, as a failure in one task may visually resemble a failure in another. To mitigate this, we treat retrieval as a constrained optimization problem. We first define the task-specific subset $M = \{v_k; M_k\} \subset M$ where $M_k = \{v_k; M_k\} \subset M$. Formally, the retrieval of the K nearest neighbors is defined as finding an optimal subset M of cardinality K that maximizes the cumulative semantic alignment with the query q_t :

$$N_K(q_t) = \arg \max_{\substack{S \subset M \\ |S|=K}} \sum_{v_k \in S} q_t^T v_k \quad (5)$$

Here, $|S|$ denotes the cardinality of the subset. This formulation ensures that our retrieval satisfies the task constraint, effectively reducing the search space complexity while guaranteeing that the retrieved metadata are relevant to the ongoing task. The retrieval result obtained by our framework is a grounded description $G(y_{sym})$ functions as a cognitive failure determination. We define a similarity score S_t as the maximum cosine similarity in the retrieved set:

$$S_t = \max_{v_k \in N_K} (q_t^T v_k) \quad (6)$$

A failure is detected if S_t exceeds a sensitivity threshold:

$$\text{State}_t = \begin{cases} 1; & \text{if } S_t > \tau \\ 0; & \text{otherwise} \end{cases} \quad (7)$$

where 1 represents failure and 0 represents normal operation. Upon triggering ($\text{State}_t = 1$), the framework issues an immediate termination signal to the robot controller. This detection mechanism bypasses the heavy inference of VLMs, providing a rapid response to prevent hardware damage.

3.4 Failure Report Generation and Utilization

Another role of RoboFailRing is to function as a consultant that provides grounded failure report to a Vision-Language Model for reasoning.

Once a failure is intercepted, the framework leverages the neuro-symbolic grounding mechanism to generate a failure report for VLMs. Instead of feeding the VLM raw images with a generic prompt which often leads to hallucination, we construct a structured spatio-temporal report of

$$R = \left\{ \left(v_k; G(y_{sym}^{(k)}); I_{start}^{(k)}; I_{end}^{(k)} \right) \mid v_k \in N_K \right\} \quad (8)$$

Here, each tuple encapsulates the essential components required for VLMs, namely the task identifier v_k , the grounded failure description $G(y_{sym})$, the camera viewpoint $I_{start}; I_{end}$. This structured design is rigorously formulated to stimulate comparative reasoning by moving beyond static image analysis.

By pairing the initial affordance I_{start} with the failure state I_{end} , we enable VLMs to track the trajectory of deviation. This allows them to analyze how the state drifted from its intended path. To further constrain the reasoning space, the explicit inclusion of task and viewpoint establishes a semantic boundary that mitigates hallucinations regarding spatial perspectives or goals, while the grounded description $G(y_{sym})$ functions as a cognitive core, precisely directing the VLM's attention to the specific failure modes (See Appendix B for details).

4 Experiments

In this section, we evaluate the failure detection effectiveness of RoboFailRing, its detection timeliness, and its impact on VLMs reasoning accuracy. We conduct these evaluations across two distinct large-scale datasets covering 81 robotic manipulation tasks, including in-distribution and out-of-distribution settings and diverse simulation environments. In addition, we perform evaluations on real-world systems. Lastly, we present ablation studies for the core components and analyze the framework's latency. Our experiments are designed to answer three core questions:

- Q1: Does RoboFailRing effectively prevent failures in manipulation tasks, even when the tasks are out of domain?
- Q2: Does failure memory retrieval enable the timely detection of failures, and specifically how much time does it save?
- Q3: To what extent can the framework's generated failure reports enhance the reasoning accuracy of VLMs, and what is the magnitude of this improvement?

Benchmark	Threshold	Recall (Detection Rate %)*						False Positive Rate (%)#					Avg.
		Dataset 1	Dataset 2	Dataset 3	Dataset 4	Dataset 5	Avg.	Dataset 1	Dataset 2	Dataset 3	Dataset 4	Dataset 5	
In-Distribution (FailGen Test Set Duan et al., 2025)	0.85	92.11	93.01	90.95	93.05	91.05	92.03	15.64	16.89	13.21	17.10	14.50	15.47
	0.90	90.97	89.54	89.80	89.90	88.98	89.84	9.64	10.08	8.08	11.79	7.90	9.50
	0.92	88.63	86.95	85.51	87.53	86.52	87.03	5.36	7.12	3.79	6.23	4.08	5.32
Out-of-Distribution (ManiSkill-failgen wpumacay, 2024)	0.85	69.68	71.98	67.17	70.83	68.72	69.68	1.13	2.44	2.28	3.00	1.42	2.05
	0.90	84.56	86.42	82.33	86.15	85.02	84.90	18.78	20.15	16.94	19.34	17.88	18.62
	0.92	83.12	83.74	80.25	83.56	82.09	82.55	12.98	14.33	10.59	14.65	11.35	12.78
	0.95	80.88	81.41	79.64	80.12	79.95	80.40	9.22	9.89	6.90	9.75	7.12	8.58
	0.95	62.15	65.55	64.32	62.90	63.18	63.62	4.25	5.98	5.14	7.11	5.45	5.59

Table 1: Effectiveness of failure detection Higher is better for Recall, and lower is better for False Positive Rate. Bold values indicate the best performance within the same benchmark. Gray row is considered the optimal threshold. Different datasets use different random seeds.

Datasets	Metrics	Avg. Value (%)
In-Distribution (FailGen Test Set)	Average Detection Progress	44.55
	Median Detection Progress	41.33
	Temporal Saving Ratio T_s	55.45
Out-of-Distribution (ManiSkill-failgen)	Average Detection Progress	55.59
	Median Detection Progress	52.36
	Temporal Saving Ratio T_s	44.41

Table 2: Time-to-detection efficiency. Higher is better for Temporal Saving Ratio and lower is better for Average Detection Progress.

4.1 Experimental Setup

4.1.1 Datasets

The retrieval memory base is constructed exclusively using the FailGen dataset. This dataset provides high-fidelity, procedurally generated failure trajectories across diverse manipulation tasks (e.g. basketball_in_hoop, open_microwave etc.), serving as the main memory for our framework. Specifically, the failure memory we use contains 5,137 failure trajectories.

4.1.2 Evaluation Benchmarks

To verify performance across different distributions, we evaluate on two distinct benchmarks comprising over 6,000 failure trajectories and covering 81 distinct robotic manipulation tasks (See Appendix E.1 and E.2 for details). We first establish an in-distribution test set using FailGen with different random seeds, enabling us to evaluate the framework's ability to handle varying object spatial configurations within the same tasks and rendering environment.

To assess cross-domain generalization, we use the ManiSkill-failgen dataset as the out-of-distribution test set. ManiSkill-failgen introduces different object assets, visual textures, and physics parameters compared to FailGen. Evaluating on this disjoint domain allows us to determine whether our failure memory have successfully captured universal failure semantics rather than merely

overfitting to specific simulator artifacts.

4.1.3 Evaluation Metrics

We propose two aspects of key metrics to assess RoboFailRing performance.

Binary Failure Detection. We treat failure detection as a binary classification problem. True Positives (TP) denote the number of correctly identified failure trajectories, and False Negatives (FN) denote the missed failure trajectories. Similarly, False Positives (FP) represents false alarms on successful trajectories, while True Negatives (TN) denotes correctly ignored successes. We use

Recall: $R = \frac{TP}{TP + FN}$ measures the proportion of ground-truth failures that are successfully detected. And **False Positive Rate** $F = \frac{FP}{FP + TN}$ evaluates the frequency of false interruptions during normal execution.

Time-to-Detection Efficiency. Beyond binary classification, the timeliness of detection is paramount for damage prevention. For a given failure trajectory of total duration L frames, let t_{det} be the frame index where the framework first triggers the alarm. We define the **Temporal Saving Ratio** (T_s) as:

$$T_s = 1 - \frac{t_{det}}{L} \quad 100\% \quad (9)$$

A higher T_s indicates that the framework intercepts the failure earlier in the trajectory, providing a larger safety margin for the robot to terminate before irreversible consequences occur.

4.2 Experimental Results

4.2.1 Effectiveness of Failure Detection

We first evaluate the effectiveness of the retrieval mechanism to answer Q1. As shown in Table 1, we evaluate the average Recall and average False Positive Rate of RoboFailRing under different similarity thresholds. For both the In-Distribution dataset

Figure 3: Distribution of detection progress on different datasets Left is In-Distribution (FailGen Test Set). Right is Out-of-Distribution (ManiSkill-failgen).

Figure 4: Real-world deployment and the reasoning performance of RoboFailRing-assisted VLMs.

(FailGen Test Set) and the Out-of-Distribution dataset (ManiSkill-failgen), the threshold = 0.92 achieves the best balance. Specifically, the results are an average Recall of 87.03% and an average False Positive Rate of 5.32% on FailGen, and an average Recall of 80.40% and an average False Positive Rate of 8.58% on ManiSkill .

4.2.2 Time-to-Detection Efficiency

Next, we evaluate the detection time of RoboFailRing to answer Q2. As shown in Table 2, we evaluate the average detection progress, median detection progress, and temporal saving ratio in RoboFailRing. Our framework consistently identifies anomalies in the intermediate stages of robotic manipulation on different datasets. Specifically, the framework achieves an average temporal saving ratio of $T_s = 55:45\%$ on FailGen and $T_s = 44:41\%$ on ManiSkill . This means that, in a circumstance where the manipulation task requires 10 seconds to complete the full failure process, the robot terminates at approximately 4 or 5 seconds. Additionally, as shown in Figure 3, the numerical distribution of detection progress ratios is not uniform. Approximately

70% of failures are detected at an early stage of the manipulation task, while the remaining 25% are sparsely distributed across different time points. It is noteworthy that about 5% of failures are not detected until the final stage of the manipulation task, and this proportion is higher on out-of-distribution datasets.

4.2.3 VLM Reasoning Improvement

We assess the impact of our structured failure report on VLMs reasoning to answer Q3. To bridge the Sim-to-Real gap, we deploy RoboFailRing on the real-world system shown in Figure 4 (See Appendix E.3 for details). We manually replicate in real-world systems, as closely as possible, the 7 failure modes previously defined in simulation, and we let the VLM reason about these failure modes with and without RoboFailRing support. If the VLM's inferred failure mode matches the ground-truth, the reasoning is counted as correct. We use Gemini-2.5-flash as a representative VLM for the experiments, and each failure mode undergoes 100 reasoning attempts. As shown in the right panel of Figure 4, the average reasoning accuracy of VLMs increases by 35%.

Ablation Aspect	Variant	Detection Recall (%)	Reasoning Acc. (%)	Latency (ms)
Grounding	raw symbolic labels	—	32.04	—
	neuro-symbolic (Ours)	—	68.14	—
View Independence	single view (Front-only)	41.62	—	—
	single view (Overhead-only)	56.18	—	—
	single view (Wrist-only)	74.12	—	—
	multi-view fusion (Ours)	88.63	—	—
Visual Encoder	ViT-L/14	90.30	—	37
	ViT-B/32	88.63	—	14

Table 3: Ablation studies of key components in RoboFailRing. Recall is failure detection rate, Reasoning Acc. is VLM failure reasoning accuracy, Latency is average detection completion time. “—” indicates the metric is not applicable to that ablation dimension.

Component	Operation	Time Cost (ms)
Detection Part (Cascaded)	image processing	4
	CLIP encoding (ViT-B/32)	10
	failure memory retrieval	1
	Total Detection Latency	15
Reasoning Part (Asynchronous)	VLM report generation	3000

Table 4: Latency Breakdown of RoboFailRing.

4.2.4 Ablation Studies

To validate the contribution of each core component, we conduct ablation experiments on the FailGen test set. Results are summarized in Table 3.

Neuro-symbolic grounding. As shown in the first part of Table 3, replacing our neuro-symbolic grounding mechanism with raw symbolic labels fed directly to the VLM causes reasoning accuracy to drop sharply from 68.14% to 32.04%. The VLM struggles to map abstract symbols to the specific visual context without the report provided by our neuro-symbolic grounding mechanism, which translates symbols into descriptive natural language prompts.

Multi-view independence. The second part highlights the necessity of our multi-view strategy. Relying on a single view causes a significant drop (Front-only: 41.62%) in Recall due to occlusions inherent in manipulation tasks. Our fusion mechanism effectively utilizes complementary information from different views and is essential for robust failure detection.

Visual encoder sensitivity. In the third part, we compared different visual backbones. Substituting ViT-B/32 with the larger ViT-L/14 yields only a marginal recall improvement of +1.67%, while nearly tripling the detection completion time from 14ms to 37ms. Since latency is the primary

constraint of our deployment scenario, we select ViT-B/32 as the default for its superior efficiency and accuracy tradeoff.

4.2.5 Latency Analysis

A critical requirement for deployment in physical robotic systems is that the failure detection loop must operate within the real-time control budget. Table 4 provides a component-level breakdown of the framework latency, measured on a local server (See Appendix A for details). The total detection latency is approximately 15ms, allowing the system to run at 66Hz. This is well within the requirements for robotic manipulation control loops. The computationally expensive VLM generation runs asynchronously and is only triggered after the robot has already been safely stopped by the detection loop, thus it does not hinder real-time safety. In addition, we conduct an experimental evaluation of how detection performance and runtime scale as the failure memory grows. The results show that, with current approximately 5k trajectories dataset, the retrieval time is around 1 ms. Even when scaling up to 50k trajectories, the retrieval time remains below 10 ms. The main bottleneck is image encoding, which is still sufficiently fast for real-time detection.

5 Conclusion

In this paper, we introduce RoboFailRing, an efficient and interpretable framework for robotic manipulation that builds a language-grounded failure memory that enables rapid failure detection and supports accurate VLM reasoning. Through large-scale simulation and real-world evaluation, it achieves high detection accuracy, reduces detection time, and improves reasoning accuracy. This demonstrates the value of our framework as a core component for more reliable robotic systems.

Limitations

RoboFailRing focuses on visually grounded, outcome-oriented failure states, and its dual-frame representation may not capture failure modes defined primarily by execution dynamics whose diagnostic signal lies in temporal evolution rather than terminal appearance. Furthermore, while our grounded failure reports improve VLM reasoning accuracy, translating the reasoned repair strategy into closed-loop recovery execution involves additional challenges in motion replanning and controller design that are outside the scope of this work, and we identify an end-to-end detect, reason, and recover pipeline as an important direction for future work. Finally, the current evaluation is limited to single-arm manipulation benchmarks, extending the framework to dual-arm or mobile-base configurations introduces new challenges that remain to be addressed.

Acknowledgments

This work is partially supported by Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China JYB2025DXM103, Key Research and Development Program of Zhejiang Province (Grant No: 2025C01012), Mobile Information Networks National Science and Technology Major Project 2026ZD1306000, and NSFC under Grants No. 92467301, 62293511.

References

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ori Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanjiv, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huang Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. 2023. [Rt-1: Robotics transformer for real-world control at scale](#). In *Proceedings of Robotics: Science and Systems*

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchel, Russ Tedrake, and Shuran Song. 2025. Diffusion policy: Visuomotor

policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704.

Muhayy Ud Din, Waseem Akram, Lyes Saad Saoud, Jan Rosell, and Irfan Hussain. 2025. [Vision language action models in robotic manipulation: A systematic review](#). Preprint, arXiv:2507.10672.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2026. [The faiss library](#). *IEEE Transactions on Big Data*, 12(2):346–361.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. 2023. [Palm-e: an embodied multimodal language model](#). In *Proceedings of the 40th International Conference on Machine Learning*

Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. 2023. [Vision-language models as success detectors](#). *Proceedings of the 2nd Conference on Lifelong Learning Agents*, pages 120–136.

Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, and Yijie Guo. 2025. [Aha: A vision-language-model for detecting and reasoning over failures in robotic manipulation](#). *The 13th International Conference on Learning Representations*

Jiafei Duan, Wentao Yuan, Wilbert Pumacay, Yi Ru Wang, Kiana Ehsani, Dieter Fox, and Ranjay Krishna. 2024. [Manipulate-anything: Automating real-world robots using vision-language models](#). *8th Annual Conference on Robot Learning*, pages 5326–5350.

Roya Firoozi, Johnathan Tucker, Stephen Tian, Anirudha Majumdar, Jiankai Sun, Weiyu Liu, Yuke Zhu, Shuran Song, Ashish Kapoor, Karol Hausman, Brian Ichter, Danny Driess, Jiajun Wu, Cewu Lu, and Mac Schwager. 2025. [Foundation models in robotics: Applications, challenges, and the future](#). *The International Journal of Robotics Research*, 44(5):701–739.

Jensen Gao, Bidipta Sarkar, Fei Xia, Ted Xiao, Jiajun Wu, Brian Ichter, Anirudha Majumdar, and Dorsa Sadigh. 2024. Physically grounded vision-language models for robotic manipulation. *2024 IEEE International Conference on Robotics and Automation*, pages 12462–12469.

Huy Ha, Pete Florence, and Shuran Song. 2023. [Scaling up and distilling down: Language-guided robot skill acquisition](#). In *7th Annual Conference on Robot Learning*

- Siyuan Huang, Haonan Chang, Yuhan Liu, Yimeng Zhu, Hao Dong, Abdeslam Boularias, Peng Gao, and Hongsheng Li. 2024. [A3vlm: Actionable articulation-aware vision language model](#). In *8th Annual Conference on Robot Learning*, pages 1675–1690.
- Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. 2023. [Voxposer: Composable 3d value maps for robotic manipulation with language models](#). In *7th Annual Conference on Robot Learning*.
- Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. 2020. [Rlbench: The robot learning benchmark & learning environment](#). *IEEE Robotics and Automation Letters*, 5(2):3019–3026.
- Byeonghwi Kim, Jinyeon Kim, Yuyeong Kim, Cheol-hong Min, and Jonghyun Choi. 2023. [Context-aware planning and environment-aware memory for instruction following embodied agents](#). In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10936–10946.
- Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. [Code as policies: Language model programs for embodied control](#). In *2023 IEEE International Conference on Robotics and Automation*, pages 9493–9500.
- Fangchen Liu, Kuan Fang, Pieter Abbeel, and Sergey Levine. 2024. [Moka: Open-vocabulary robotic manipulation through mark-based visual prompting](#). In *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*.
- Zeyi Liu, Arpit Bahety, and Shuran Song. 2023. [Reflect: Summarizing robot experiences for failure explanation and correction](#). In *7th Annual Conference on Robot Learning*.
- Zixian Liu, Mingtong Zhang, and Yunzhu Li. 2025. [Kuda: Keypoints to unify dynamics learning and visual prompting for open-vocabulary robotic manipulation](#). In *2025 IEEE International Conference on Robotics and Automation*, pages 10561–10569.
- Yecheng Jason Ma, Joey Hejna, Chuyuan Fu, Dhruv Shah, Jacky Liang, Zhuo Xu, Sean Kirmani, Peng Xu, Danny Driess, Ted Xiao, Osbert Bastani, Dinesh Jayaraman, Wenhao Yu, Tingnan Zhang, Dorsa Sadigh, and Fei Xia. 2025. [Vision language models are in-context value learners](#). In *The 13th International Conference on Learning Representations*.
- Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. 2023. [Vip: Towards universal visual reward and representation via value-implicit pre-training](#). In *The 11th International Conference on Learning Representations*.
- Muhammad A Muttaqien, Tomohiro Motoda, Ryo Hanai, and Yukiyasu Domae. 2025. [Visual prompting for robotic manipulation with annotation-guided pick-and-place using act](#). In *2025 IEEE 21st International Conference on Automation Science and Engineering*, pages 2642–2649.
- Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. 2024. [Octo: An open-source generalist robot policy](#). In *Proceedings of Robotics: Science and Systems*.
- Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. [Learning transferable visual models from natural language supervision](#). In *Proceedings of the 38th International Conference on Machine Learning*, pages 8748–8763.
- Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. 2024. [Gensim: Generating robotic simulation tasks via large language models](#). In *The 12th International Conference on Learning Representations*.
- wpumacay. 2024. [maniskill-failgen: A implementation of failgen in maniskill](#).
- Chenduo Ying, Linkang Du, Peng Cheng, and Yuanchao Shu. 2025. [Roboinspector: Unveiling the unreliability of policy code for llm-enabled robotic manipulation](#). *Preprint*, arXiv:2508.21378.
- Minjong Yoo, Jinwoo Jang, Wei-Jin Park, and Honguk Woo. 2024. [Exploratory retrieval-augmented planning for continual embodied instruction following](#). In *The 39th Annual Conference on Neural Information Processing Systems*, pages 67034–67060.
- Samson Yu, Kelvin Lin, Anxing Xiao, Jiafei Duan, and Harold Soh. 2024. [Octopi: Object property reasoning with large tactile-language models](#). In *Proceedings of Robotics: Science and Systems*.
- Wentao Yuan, Jiafei Duan, Valts Blukis, Wilbert Pumacay, Ranjay Krishna, Adithyavairavan Murali, Arsalan Mousavian, and Dieter Fox. 2024. [Robotpoint: A vision-language model for spatial affordance prediction in robotics](#). In *8th Conference on Robot Learning*, volume 270, pages 4005–4020.
- Yi Zhang, Che Liu, Xiancong Ren, Hanchu Ni, Shuai Zhang, Zeyuan Ding, Jiayu Hu, Hanzhe Shan, Zhenwei Niu, Zhaoyang Liu, Shuang Liu, Yue Zhao, Junbo Qi, Qinfan Zhang, Dengjie Li, Yidong Wang, Jiachen Luo, Yong Dai, Zenglin Xu, Bin Shen, Qifan Wang, Jian Tang, and Xiaozhu Ju. 2025. [Pelican-vl 1.0: A foundation brain model for embodied intelligence](#). *Preprint*, arXiv:2511.00108.

Yue Zheng, Yuhao Chen, Bin Qian, Xiufang Shi, Yuan-chao Shu, and Jiming Chen. 2025. [A review on edge large language models: Design, execution, and applications](#). *ACM Computing Surveys*, 57(8):1–35.

Yichen Zhu, Zhicai Ou, Xiaofeng Mou, and Jian Tang. 2024. [Retrieval-augmented embodied agents](#). In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17985–17995.

Appendix

A Platform

Our experiments in simulation are conducted on a server running a 64-bit Ubuntu 20.04.6LTS system with 48 AMD(R) Epyc 7402 @2.8GHz 24-core processors, 128GB memory, and four Nvidia RTX3090 GPUs, each with 24GB memory. The experiments are performed using the Python3.10.18. Our experiments on the real-world systems are conducted on a 6-DoF myCobot robot arm with myCobot gripper from Elephant Robotics. In addition, we employ a compact Femto Bolt camera equipped with multi-modal depth and RGB sensors. It is manufactured by Orbbec.

B Prompt Template

Prompt for VLM reasoning

"role": "system", "content":

"You are an expert robotic failure analyst. Use the provided Failure Report and Reference Images to diagnose the Current Situation."

"role": "user", "content":

"Robot stopped now because the current scene is similar to the failure memory. I have retrieved a similar historical failure case from memory. Details of Failure Report as follows:

```
report = {  
  "current_task": task_name,  
  "failure": grounded failure  
  description,  
  "view": viewpoint,  
  "similarity_score": f"r_score:.2f",  
}
```

"role": "user", "content":

"Below are the reference images from this historical failure memory:",
"Reference Start Frame (Task Context):",

```
"reference_start_img":  
  os.path.abspath(start_img_path),  
"Reference End Frame (Failure State):",  
"reference_end_img":  
  os.path.abspath(end_img_path),
```

"role": "user", "content":

"Now, look at the current robot observation:",

"Current Frame:",

"current_img":

```
os.path.abspath(current_img_path),
```

"role": "user", "content":

"Based on this report, please reasoning why it failed and how should it be repaired? You tasks are:

1. Compare the Current Observation with the Reference End Frame.
2. Confirm if the failure mode described in the report matches the visual evidence.
3. Provide a reasoning for the failure cause.
4. Suggest a concrete repair action."

"role": "assistant", "content":

"Got it. Here's the analysis of the current situation:"

Example output of VLM

"role": "assistant", "content":

"1. Comparison of Current Observation with Reference End Frame:

The Current Robot Observation is visually identical to the Reference End Frame. In both images, the robot's gripper is positioned horizontally, parallel to the ground, and is very close to the microwave's front panel. The gripper's opening is facing downwards or slightly inwards, making it impossible to grasp the vertical microwave handle. The robot arm is bent, indicating it has attempted to reach the target but is in an incorrect final pose.

2. Confirmation of Failure Mode:

Yes, the failure mode described in the re-

port ("The failure was caused by incorrect gripper rotation along the Z-axis at the initial approach stage (Waypoint 1)") matches the visual evidence. The gripper is clearly misoriented. For opening a microwave handle, the gripper typically needs to be oriented vertically to grasp the handle. In the observed state, the gripper is horizontally oriented, which is an incorrect rotation. Assuming "Z-axis" refers to the gripper's forward axis (tool-Z), then rotation along this axis is a roll motion, which is precisely what appears to be incorrect here. The gripper is "rolled" 90 degrees from its expected upright orientation.

3. Reasoning for the Failure Cause:

The task is to "open_microwave". This requires the robot to grasp the microwave handle, which is typically a vertical bar. For a successful grasp, the gripper must approach with an orientation that allows its fingers to encompass the handle. The observed failure state shows the gripper oriented horizontally, with its opening facing downwards. This incorrect orientation, likely a roll error (rotation around the gripper's forward axis, which the report refers to as Z-axis rotation), prevents the gripper from aligning with and grasping the vertical handle. The error occurring at an "initial approach stage (Waypoint 1)" means this fundamental orientation mistake was made early in the trajectory, leading to the observed unrecoverable final approach pose.

4. Suggested Concrete Recovery Action:

1. Retract: Move the gripper back to a safe, pre-defined waypoint (e.g., the initial approach waypoint or slightly further back from the microwave).
2. Correct Gripper Orientation: Rotate the gripper by approximately 90 degrees around its end-effector's forward (Z) axis (i.e., perform a roll adjustment) to orient it vertically.
3. Re-approach and Grasp: Re-execute the approach trajectory towards the microwave handle with the corrected gripper orientation, and then attempt the grasping action."

C Neuro-Symbolic Grounding Rules

To enable interpretable neuro-symbolic grounding, we map discrete symbolic failure labels to human-readable natural language semantics. Specifically, we define three mapping groups: (1) error-type grounding, which explains the physical cause of failure; (2) axis grounding, which specifies the spatial dimension involved; and (3) execution-stage grounding, which situates the failure temporally within the task procedure. The complete mapping rules are summarized in Table 5.

Symbolic Token	Grounded Natural Language Description
(a) Error-Type Grounding	
rotation	incorrect gripper rotation
no_rotation	gripper translates to the correct pose but fails to rotate
translation	positional deviation of the end-effector
slip	object slippage from the gripper
grasp	gripper reaches target pose but fails to close
wrong_sequence	robot executes actions in an incorrect order
wrong_object	robot operates on an unintended object
(b) Axis Grounding	
_x	along the X-axis
_y	along the Y-axis
_z	along the Z-axis
(c) Execution-Stage Grounding	
wp1	initial stage (Waypoint 1)
wp2	interaction / grasping stage (Waypoint 2)
wp3	manipulation / movement stage (Waypoint 3)
wp4	near-goal placement stage (Waypoint 4)
wp5	final completion / retraction stage (Waypoint 5)
(d) Episode Grounding	
episode	ignored for linguistic grounding

Table 5: Unified symbolic-to-language grounding table.

Algorithm 1 Pseudo code for constructing failure memory

Require: Dataset \mathcal{D} , CLIP encoder f , camera views \mathcal{V}

Ensure: Faiss index \mathcal{I} and metadata list \mathcal{M}

- 1: Initialize empty index \mathcal{I} and metadata \mathcal{M}
- 2: **for** each failure trajectory $(task, failure)$ in \mathcal{D} **do**
- 3: **for** each view $v \in \mathcal{V}$ **do**
- 4: Extract ordered image sequence \mathcal{S}_v
- 5: **if** $|\mathcal{S}_v| < 2$ **then continue**
- 6: **end if**
- 7: $I_s \leftarrow$ initial frame of \mathcal{S}_v ; $I_e \leftarrow$ end frame of \mathcal{S}_v
- 8: Encode and normalize CLIP embeddings:

$$z_s = \frac{f(I_s)}{\|f(I_s)\|}, \quad z_e = \frac{f(I_e)}{\|f(I_e)\|}$$

- 9: Concatenate start/end representations:

$$z = [z_s \parallel z_e]$$

- 10: Normalize joint embedding: $z = \frac{z}{\|z\|}$
- 11: Add z to index \mathcal{I}
- 12: Append $(task, failure, v, I_s, I_e)$ to \mathcal{M}
- 13: **end for**
- 14: **end for**
- 15: Save \mathcal{I} and \mathcal{M}

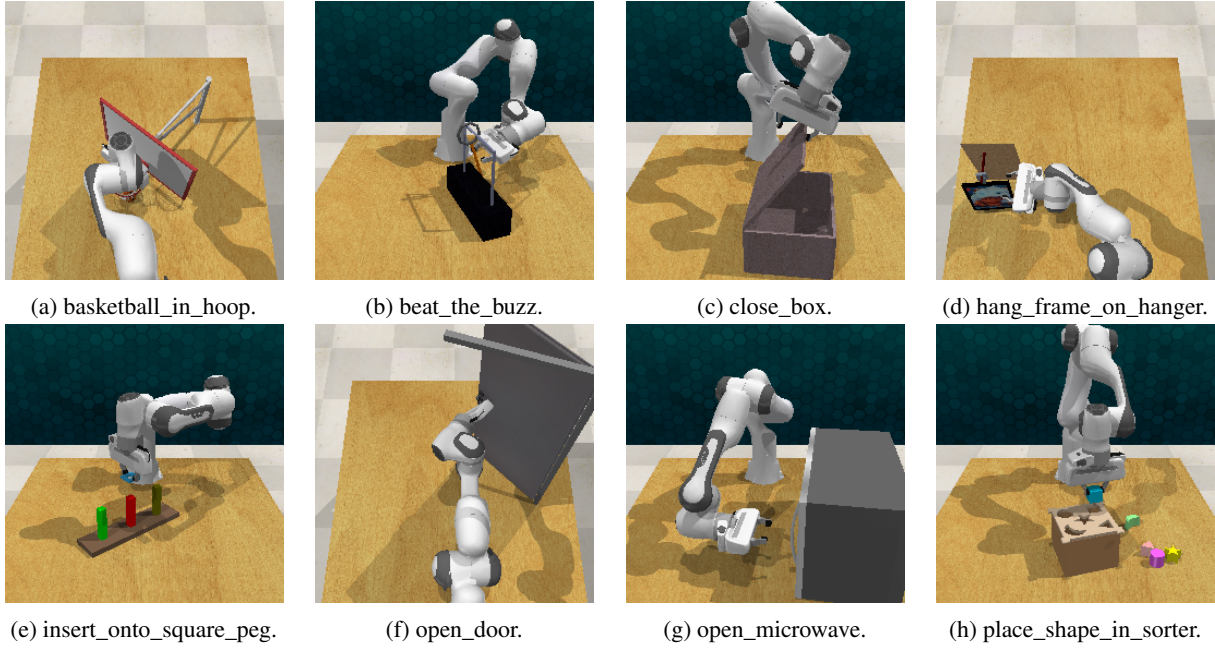


Figure 5: Eight Experiment tasks in FailGen.

D Failed Memory Construction

We construct the failure memory by extracting embeddings from the keyframes of each failure trajectory (Algorithm 1), concatenating them into joint vectors, and storing them with metadata in Faiss.

E Experiment scene

E.1 Experiment tasks in FailGen

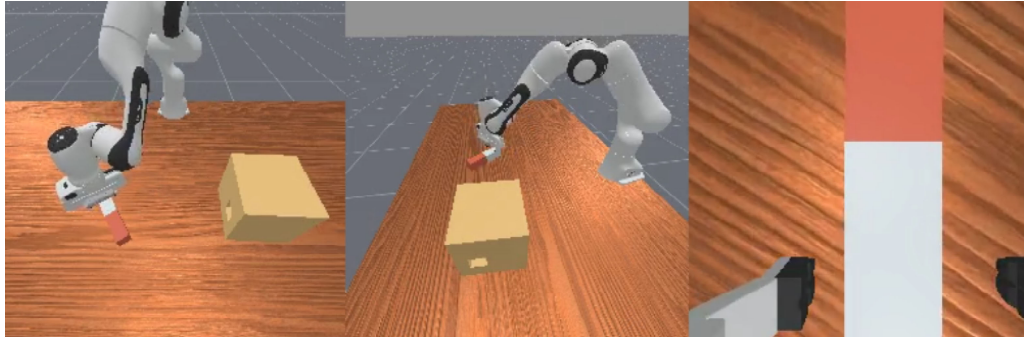
FailGen contains a total of 78 robotic manipulation tasks, and a selection of these tasks is presented in Figure 5.

E.2 Experiment tasks in ManiSkill

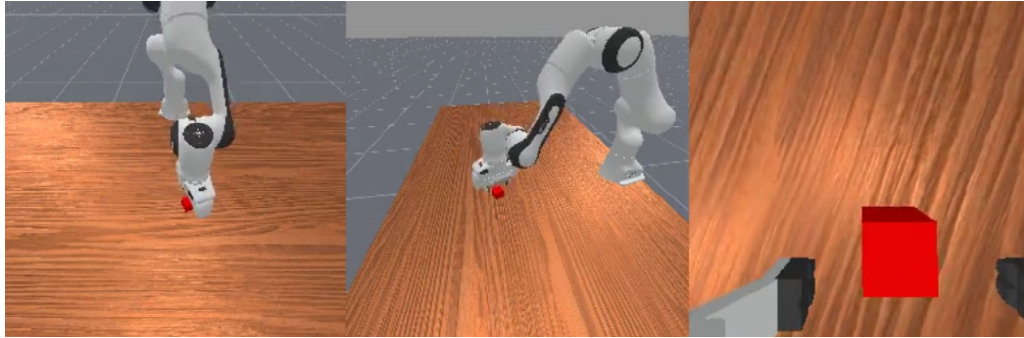
ManiSkill includes a total of three robotic manipulation tasks, all of which are presented here in Figure 6.

E.3 Experiment scene in real-world system

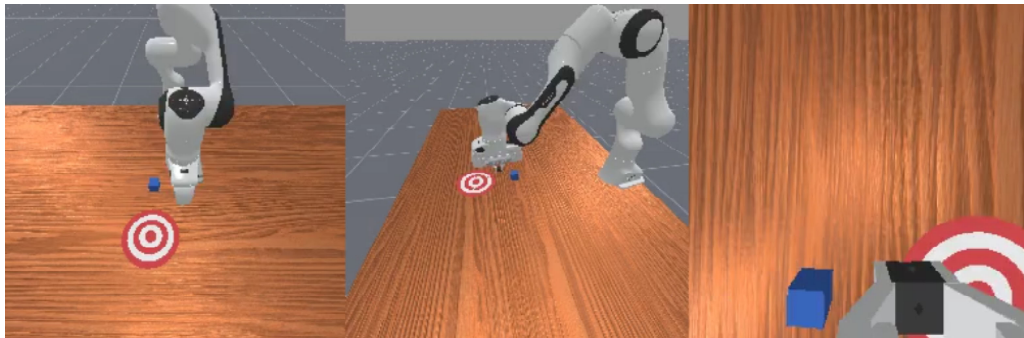
We manually replicate the seven failure modes defined in the simulation dataset as much as possible in the real-world system, and we display all of them in Figure 7.



(a) FailPegInsertionSide.



(b) FailPickCube.



(c) FailPushCube.

Figure 6: Three Experiment tasks in ManiSkill.



(a) No_Grasp.



(b) No_Rotation.



(c) Rotation.

(d) Slip.

(e) Translation.

(f) Wrong_Object.

(g) Wrong_Sequence.

Figure 7: Seven different failure modes in the real-world system.