LIBERO-RECOVER: BEYOND TASK SUCCESS TOWARDS FAILURE RECOVERY IN ROBOTIC MANIPULATION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) or World Action (WAM) models have recently demonstrated remarkable performance in robotic manipulation. On LIBERO, SOTA method have achieved nearly 100% success rates, seemingly suggesting that the models are ready for deployment in real world. However, near perfect performance on existing benchmarks can be misleading: success under ideal conditions does not imply real world robustness. Existing benchmarks primarily evaluate task completion from predefined initial states, while real world interactions inevitably involve failures such as failed grasps, collisions, and unintended object movements. A robot must therefore not only execute tasks successfully, but also recognize and recover from failures to continue the task. Yet this capability remains largely unmeasured, revealing a critical gap between benchmark performance and real world reliability. To address this gap, we introduce LIBERO-Recover Benchmark, a large scale benchmark for failure recovery in robotic manipulation. Built upon LIBERO, we collect real execution failures from SOTA embodied models and construct 2,000+ scenarios across four recovery levels: (1) Action Retry, (2) Action Adaptation, (3) Object State Recovery, and (4) Environmental Recovery. We evaluate four core capabilities: spatial understanding, object structure reasoning, interaction understanding, and topological reasoning. As the first large-scale benchmark for embodied failure recovery, LIBERO-Recover shifts evaluation from *Can the robot succeed?"* to *Can the robot recover after failure?"*, promoting robust and generalizable embodied agents. The project will be available.

1 INTRODUCTION

VLAs and WAMs have achieved remarkable progress in robotic manipulation. On early benchmarks such as LIBERO Liu et al. (2023), SOTA models have approached 100% success rates, with some recent works even omitting LIBERO results Liu et al. (2023), as its tasks are no longer sufficiently challenging to differentiate model capabilities. Yet, from LIBERO Liu et al. (2023) and SimplerEnv Li et al. (2024) to RoboTwin2.0 Chen et al. (2025) and RoboCasa Nasiriany et al. (2024), the prevailing evaluation paradigm remains largely unchanged: models start from predefined initial states and are evaluated on single-attempt task completion. This setting assumes largely idealized execution, whereas real-world manipulation inevitably involves failed grasps, collisions, and unintended object displacement. Robust robots must therefore not only complete tasks, but also recognize failures, recover disrupted states, replan, and resume execution.

Recent benchmarks have extended LIBERO toward more challenging and realistic settings. LIBERO-Pro Zhou et al. (2025) introduces more complex task variations, LIBERO-Plus Fei et al. (2025) evaluates robustness across multiple dimensions, LIBERO-X Wang et al. (2026) considers variations in task and environmental conditions, and LIBERO-Safety Cui et al. (2026) focuses on safety-critical failures and undesired interactions.

However, these benchmarks primarily assess generalization under predefined perturbations or challenging conditions, rather than recovery from naturally occurring execution failures. Failure recognition, state recovery, and post-failure replanning therefore remain insufficiently evaluated. To address

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

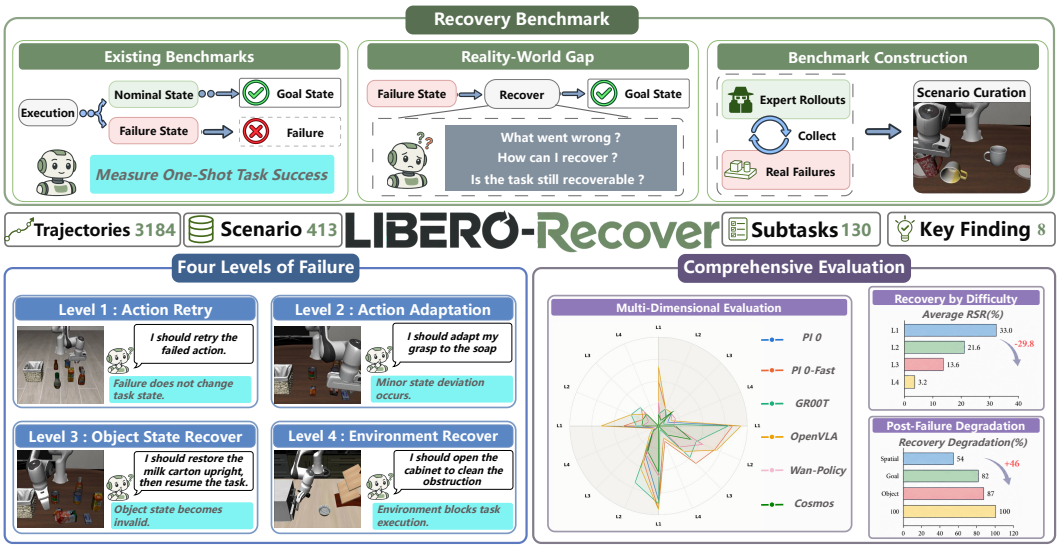


Figure 1: **Overview of LIBERO-Recover.** LIBERO-Recover collects failure scenarios from SOTA models and expert recovery trajectories via human teleoperation. The plots summarize recovery performance across difficulty levels and performance degradation after failure.

this gap, we propose LIBERO-Recover, a benchmark that collects failure states and their consequences from embodied model executions to evaluate failure recovery. Unlike existing benchmarks that construct test environments by manually perturbing object poses, appearances, or scene layouts, LIBERO-Recover naturally collects failure states and their consequences arising from real embodied models. Its evaluation protocol assesses four core capabilities—embodiment understanding, object structure reasoning, topological interaction reasoning, and failure recovery—across 130 subtasks and 16 evaluation dimensions. Experiments show that existing VLA models' failure recovery capabilities remain substantially limited, revealing a significant gap between high task success rates and robustness under real-world interactions. Our key contributions are summarized as follows.

- We show that LIBERO evaluates one-shot task success under idealized conditions, potentially overestimating the robustness of embodied models to real-world failures.
- We introduce **LIBERO-Recover**, a failure recovery benchmark built from naturally occurring failures during embodied model interactions, covering **4 task categories, 130 subtasks, and 16 level combinations** to assess four core recovery capabilities.
- We evaluate leading VLA models and reveal a substantial performance gap between standard task execution and failure recovery, highlighting critical limitations in failure-state understanding and recovery planning.

2 LIBERO-RECOVER BENCHMARK

2.1 PROBLEM FORMULATION

We consider a manipulation task specified by a language instruction l, with a goal condition $\mathcal{G}$. Starting from an initial state s_0, a manipulation policy π_θ generates a sequence of actions based on the robot's observations, resulting in an execution trajectory:

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_T). \tag{1}$$

Conventional Task Execution. Standard evaluation measures whether a policy completes the task from a valid initial state s_0, reaching a terminal state s_T with $\mathcal{G}(s_T, l) = 1$, without separately assessing post-failure recovery.

Failure Recovery. Our setting instead considers an execution that deviates from the intended task progression and reaches an intermediate state s_f. We refer to s_f as a *failure state* when the original



Figure 2: **Four recovery difficulty levels of LIBERO-Recover.** Each level demands progressively deeper state reasoning, from re-executing a failed action (L_1) to restoring environmental states that block task completion (L_4).

task goal is not satisfied at this point,

$$\mathcal{G}(s_f, l) = 0, \tag{2}$$

but the task remains recoverable through subsequent interaction. The key difference is that the policy can no longer simply continue the original action sequence. Instead, it must determine how the preceding failure has altered the current task configuration and generate a new sequence of actions that restores the necessary state before continuing toward the original goal. We therefore decompose the execution into a pre-failure trajectory and a recovery trajectory:

$$s_0 \to \cdots \to s_f \to s_{f+1} \to \cdots \to s_T, \tag{3}$$

where the post-failure actions constitute a recovery policy π_{rec}. A failure state is considered *recoverable* if:

$$\mathcal{G}(s_f, l) = 0 \quad \text{and} \quad \exists, \pi_{\text{rec}} \text{ s.t.} \mathcal{G}(s_T, l) = 1. \tag{4}$$

Failure recovery requires robots to understand, adapt, and recover from unexpected states.

2.2 Failure or State Variation?

Not every state change during execution constitutes a failure. We define a state s_f as a failure state when the preceding action induces a task-relevant change that invalidates the current execution plan:

$$\mathcal{F}(s_t, a_t, s_f) \iff \underbrace{s_t \xrightarrow{a_t} s_f}_{\text{execution-induced}} \wedge \underbrace{\mathcal{G}(s_f, l) = 0}_{\text{task-relevant}} \wedge \underbrace{\pi_\theta(s_f) \not\Rightarrow \mathcal{G}}_{\text{plan-invalidating}}. \tag{5}$$

Here, the last condition means that directly continuing the current execution is insufficient to complete the original task, requiring adaptation or recovery. A valid failure state must nevertheless remain recoverable,

$$\exists\, \pi_{\text{rec}} \quad \text{s.t.} \quad s_f \xrightarrow{\pi_{\text{rec}}} s_T, \quad \mathcal{G}(s_T, l) = 1. \tag{6}$$

Thus, state changes that preserve task executability are treated as normal variations, whereas execution-induced changes that invalidate the current plan and require recovery are treated as failures.

2.3 Failure Taxonomy and Difficulty Levels

We organize recoverable failures into four difficulty levels according to the corrective actions and state reasoning required to resume task execution, as illustrated in Fig. 2. The hierarchy distinguishes failures that can be resolved by retrying or adapting actions from those requiring object state restoration or the removal of environmental obstructions.

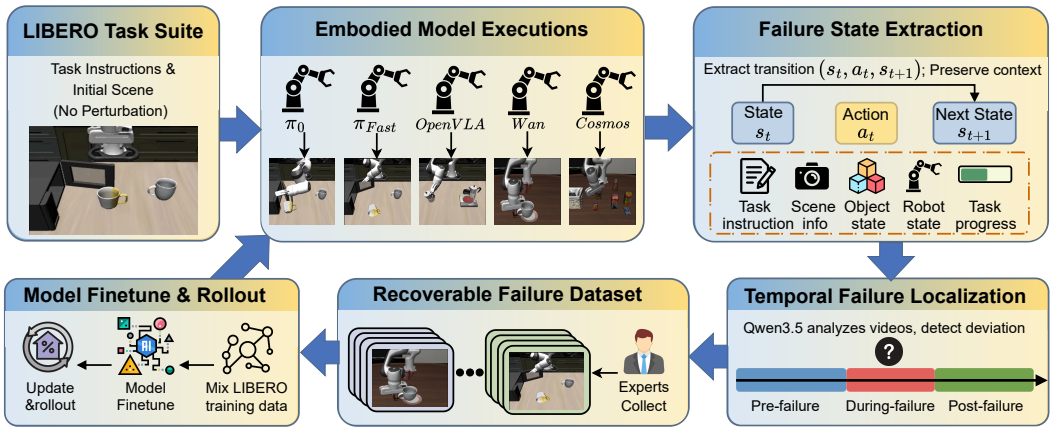


Figure 3: The construction pipeline of LIBERO-Recover.

L_1 **Action Retry**. The failure does not materially alter the task configuration, i.e., $s_f \approx s_{\text{expected}}$. Recovery therefore only requires retrying the failed action, such as reattempting a grasp when the target object remains unchanged.

L_2 **Action Adaptation**. The failure causes a limited change in the object configuration, such that $s_f \neq s_{\text{expected}}$, but the object remains directly usable. The model must adapt its action to the observed state rather than blindly repeating the failed action:

$$a' = \pi(o_f, l) \neq a_{\text{failed}}. \tag{7}$$

L_3 **Object State Recovery**. The failure substantially alters a task-relevant object, making direct task completion impossible. The model must first restore the object's required state and then resume the original task. For example, after a failed grasp displaces a bowl, the robot must first recover the bowl's pose before continuing the instructed manipulation.

L_4 **Environmental Recovery**. The failure affects an object or state that is not directly manipulated by the original task but subsequently prevents task execution. Recovery therefore requires reasoning about both task-relevant and task-irrelevant states and their interaction topology. The model must identify the environmental obstruction, restore the configuration, and resume the original task.

2.4 Recovery Scenario Construction

Scenario Definition. Rather than manually perturbing task states to synthesize failures, LIBERO-Recover grounds every scenario in a failure that actually occurred during embodied model execution. Formally, each recovery scenario is defined as

$$(\mathcal{I}, s_0, \tau_{\text{fail}}, s_f, g, r), \tag{8}$$

where $\mathcal{I}$ denotes the task instruction, s_0 the original initial state, τ_{fail} the failure trajectory, s_f the resulting failure state, g the original task goal, and r the required recovery behavior.

Construction Pipeline. As illustrated in Fig. 3, each scenario is constructed through a four-stage pipeline, with the source tasks, failure generators, and annotation models introduced stage by stage.

Stage 1: Task Execution. We start from four LIBERO task suites covering 130 subtasks, preserving their original instructions and initial configurations. Multiple embodied policies, including π_0, $\pi_{0.5}$, OpenVLA, GR00T, Wan-Policy, and Cosmos-Policy, are deployed as *failure generators*. Each execution from s_0 produces a complete failure trajectory τ_{fail}, and the simulator states are rendered into videos for subsequent temporal analysis. No objects, environment states, or initial configurations are manually perturbed, and no failures are injected during execution.

Stage 2: Failure Localization. For each execution video, we employ Qwen3.5-27B to analyze the entire execution together with the task instruction and to temporally localize where the execution deviates from the intended progression. Each video is decomposed into three stages,

$$V = V_{\text{pre}} \cup V_{\text{fail}} \cup V_{\text{post}}, \tag{9}$$

corresponding to the execution before, during, and after the failure-causing interaction. Based on the predicted temporal boundaries, we retrieve the corresponding robot simulation states and object poses directly from the simulator, and extract the localized failure scenario.

Stage 3: Failure Characterization. For each candidate failure scene, we render its execution trajectory into a video and provide the video and corresponding task instruction to Qwen3.5-27B. The model is then prompted to assess the post-failure consequences and determine the corresponding recovery type and difficulty level according to the taxonomy defined in Sec. 2.3.

Stage 4: Model Finetune & Rollout. We fine-tuned the model using manually collected recovery data from failure scenarios, and deployed it under the original LIBERO settings to collect further failure trajectories. Scenarios sampled for recovery data were excluded from the final evaluation set.

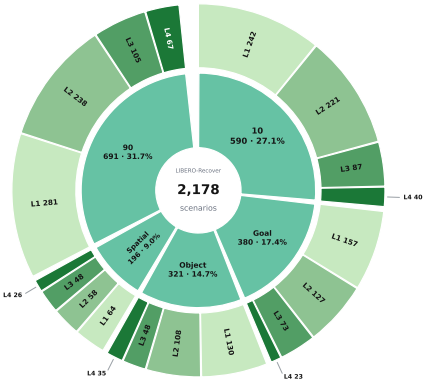


Figure 4: Distribution of LIBERO-Recover scenarios across task suites and recovery difficulty levels, including LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, LIBERO-10 (LIBERO-100) and LIBERO-90 (LIBERO-100). For detailed analysis, LIBERO-100 is further partitioned into the LIBERO-10 and LIBERO-90 subsets.

2.5 Benchmark Statistics

The resulting benchmark contains **2178** recovery scenarios derived from **130 subtasks** across **four task categories**, covering **four recovery levels** and **16 evaluation dimensions**. In total, failures are generated by **six embodied models**. The overall benchmark statistics are summarized in Fig. 4. Across the five evaluation suites, the four recovery levels exhibit an inherently imbalanced distribution, with higher-difficulty failures occurring less frequently. This distribution emerges naturally from real-world model executions, and we therefore preserve it without manual rebalancing to faithfully reflect the empirical failure distribution. Notably, the proportion of Post-Task-Failure cases increases with recovery difficulty, as models lacking recovery capability tend to repeatedly retry failed actions, often leading to progressively more severe consequences.

3 Datasets and Evaluation Metrics

3.1 LIBERO-Recover Dataset

To improve failure recovery, four human teleoperators collected 3,184 recovery trajectories across 413 failure scenarios using a SpaceMouse, uniformly sampled from the five task suites. These trajectories are used for model fine-tuning. Finetuned models are used to re-generate failure scenarios.

3.2 Evaluation Metrics

To assess models' ability to handle failures, we propose the following metrics for evaluation:

Recovery Success Rate (RSR), which measures whether a policy can successfully complete the original task after entering a failure state. For N failure scenarios, we define

$$\mathrm{RSR} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\left[\mathcal{G}(s_T^i, l_i) = 1\right], \tag{10}$$

where s_T^i denotes the terminal state of the recovered episode. A higher RSR indicates stronger failure recovery capability. To characterize the effect of recovery difficulty, we additionally report RSR_{L_1}, RSR_{L_2}, RSR_{L_3}, and RSR_{L_4} for the four recovery levels.

Recovery Degradation (RD). To quantify execution capability lost after failure, we divide each episode into three phases: *Pre-Failure*, *During-Failure*, and *Post-Failure*. Let M_{pre} and M_{post}

denote the execution performance before and after failure. We define *Recovery Degradation (RD)* as

$$\mathrm{RD} = 1 - \frac{M_{\mathrm{post}} + \epsilon}{M_{\mathrm{pre}} + \epsilon}, \tag{11}$$

where ϵ ensures numerical stability. Lower RD indicates better retention of execution capability, while higher RD indicates greater degradation.

Recovery Consistency (RC), which measures the stability of recovery across different failure states within the same task. For each task k, let R_k denote its recovery success rate across all associated failure scenarios. We define

$$\mathrm{RC} = 1 - \mathrm{Std}_k(R_k), \tag{12}$$

where $\mathrm{Std}_k(\cdot)$ denotes the standard deviation of task-level recovery success rates. Higher RC indicates more consistent recovery across failure states.

4 EXPERIMENTS

4.1 EXPERIMENTS SETUP

We evaluate six representative models: OpenVLA-OFT Kim et al. (2024), π_0-Fast Pertsch et al. (2025), GR00T-N1.5 Bjorck et al. (2025), π_0 Black et al. (2024), Wan2-Policy Wan et al. (2025), and Cosmos-Predict2-Policy Alhaija et al. (2025). VLA methods use VLM backbones to encode visual observations and language instructions, while WAM methods generate latent action representations followed by an action head. All implementations follow the official StarVLA framework Community (2026). Following LIBERO Liu et al. (2023), each task is evaluated over 10 trials with slight object-position perturbations, with a maximum rollout horizon of 520 control steps. Initial scene configurations are recorded to ensure consistent starting conditions. Additional experimental details are provided in the appendix.

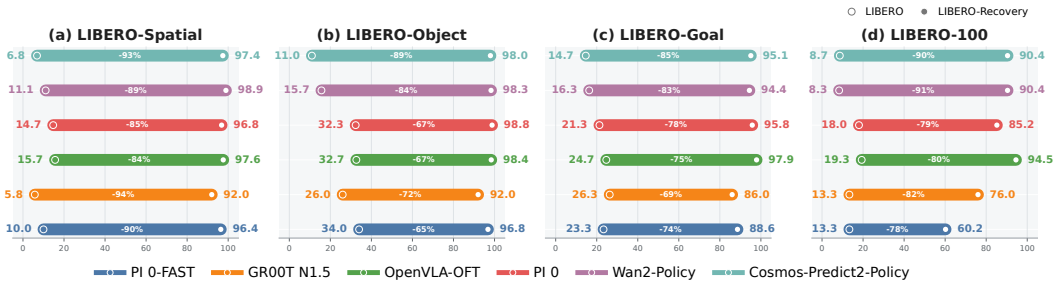


Figure 5: The performance of VLAS and WAMS on LIBERO and LIBERO-Recover.

4.2 MAIN RESULTS

4.2.1 MULTI-LEVEL EVALUATION

Key Finding 1: Success in predefined scenarios does not guarantee robust generalization to real failures. As shown in Fig. 5, performance on predefined scenarios does not reliably reflect embodied models' capabilities under real execution. All models suffer over 50% performance drops when exposed to naturally occurring failures. More importantly, performance advantages on standard benchmarks do not consistently transfer to failure scenarios. For example, Wan2-Policy outperforms GR00T-N1.5 by +14.40% on LIBERO-100, yet falls -5.0% behind it under corresponding failures. Similarly, π_0 and π_0-Fast exhibit substantial performance reversals on LIBERO-Spatial. These results demonstrate that standard benchmark rankings are poor predictors of robustness to real execution failures and recovery capability.

Key Finding 2: Existing manipulation models excel at local correction but struggle with state recovery. All models achieve substantially higher success rates on L_1/L_2 failures than on L3/L4, with performance consistently decreasing as recovery difficulty increases. This progressive degradation suggests that current manipulation models can locally adjust their immediate actions but struggle to reason about and recover the underlying task state after failure. L1 and L2 can often

Table 1: The Multi-level Evaluation on LIBERO-Recover. Darker cells indicate higher recovery success rates. Best results in each difficulty level are shown in bold. PI (PI-0), FAST (PI-0 FAST), GR00T (GR00T N1.5), Wan.(Wan2-Policy) and Cos.(Cosmos-Predict2-Policy).

	LIBERO-Spatial				LIBERO-Object				LIBERO-Goal				LIBERO-100			
Method	**L1**	**L2**	**L3**	**L4**	**L1**	**L2**	**L3**	**L4**	**L1**	**L2**	**L3**	**L4**	**L1**	**L2**	**L3**	**L4**
PI	**40.8**	11.2	13.7	6.0	**56.4**	**41.9**	24.6	8.4	50.6	21.9	5.9	2.7	**39.4**	**20.6**	10.1	0.0
FAST	29.3	13.1	6.9	3.2	49.2	41.7	**35.2**	**12.2**	**56.9**	25.8	5.6	0.0	24.5	10.8	16.2	0.0
GR00T	7.4	13.2	7.3	3.4	47.9	26.8	30.6	1.8	51.9	**41.8**	5.8	0.0	14.3	**20.6**	**17.6**	0.0
OFT	**40.8**	12.1	12.8	8.6	**56.4**	**41.9**	25.1	7.9	**56.9**	30.8	**8.9**	**3.9**	**39.4**	**20.6**	10.1	0.0
Wan.	16.7	3.6	11.8	**12.4**	9.3	33.1	20.1	0.0	34.4	20.8	4.2	2.9	10.7	7.6	13.9	0.0
Cos.	7.4	4.4	**14.2**	0.0	10.8	24.5	8.5	0.0	30.7	19.4	5.7	0.0	11.9	10.9	11.2	0.0

be addressed by adapting the next action based on visual observations, whereas L3 and L4 require structured reasoning about changes in object states, spatial relations, and interaction dependencies, as well as sufficient scene memory. The substantial gap between L2 and L3 therefore reveals a transition from *action-level correction* to *state-level recovery*, which remains a major limitation of existing manipulation models.

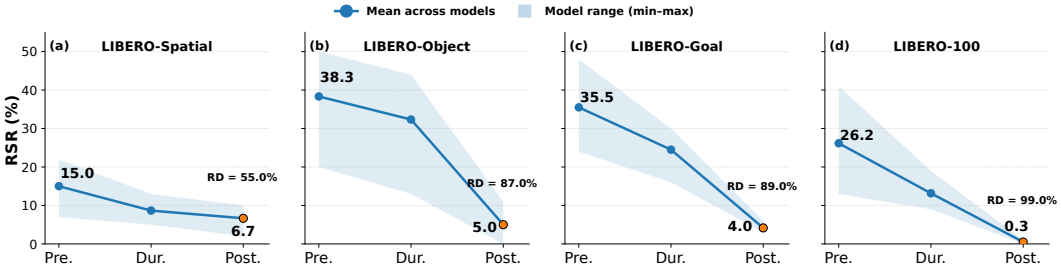


Figure 6: Temporal Manifestation of Failures (RD) in VLAs and WAMs on LIBERO-Recover.

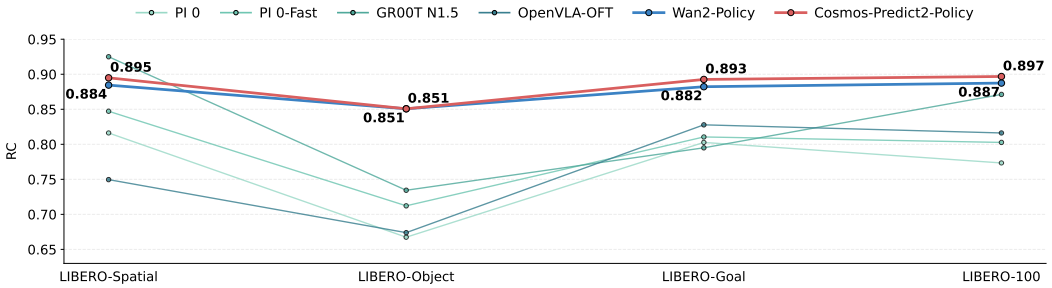


Figure 7: The **RC** performace of VLAS and WAMS on LIBERO-Recover.

4.2.2 Multi-label Evaluation

Key Finding 3: Failures induce compounding distribution shifts that progressively degrade model performance. All evaluated manipulation models exhibit substantial performance degradation after failure across all four LIBERO suites. As shown in Fig. 6, averaged across models, the success rate drops from 15.0% to 6.7% on LIBERO-Spatial, 35.5% to 4.0% on LIBERO-Goal, 38.3% to 5.0% on LIBERO-Object, and 26.2% to only 0.3% on LIBERO-100, corresponding to recovery degradation rates of 55.0%, 87.0%, 89.0%, and 99.0%, respectively. Notably, performance further decreases from the failure stage to the post-failure stage, suggesting that repeated unsuccessful interactions may progressively drive the execution into increasingly unfa-

Table 2: Performance on LIBERO and LIBERO-Recover after training with mixed LIBERO-Recover expert data.

LIBERO				
Method	LIBERO-Spatial	LIBERO-Object	LIBERO-Goal	LIBERO-100
GR00T N1.5	0.920	0.920	0.860	0.760
GR00T N1.5 (Combined)	0.931	0.905	0.882	0.770
OpenVLA-OFT	0.976	0.984	0.979	0.945
OpenVLA-OFT (Combined)	0.982	0.994	0.970	0.916
LIBERO-Recover				
Method	LIBERO-Spatial	LIBERO-Object	LIBERO-Goal	LIBERO-100
GR00T N1.5	0.057	0.260	0.263	0.133
GR00T N1.5 (Combined)	0.116	0.262	0.264	0.206
OpenVLA-OFT	0.156	0.326	0.246	0.193
OpenVLA-OFT (Combined)	0.178	0.333	0.270	0.236

miliar states. These results indicate that failure is not merely a transient disturbance, but can induce a compounding distribution shift that substantially impairs subsequent task execution.

Key Finding 4: WAM policies exhibit more stable recovery. WAM policies consistently achieve higher Recovery Consistency (RC) than conventional VLAs, with Cosmos-Predict2-Policy and Wan2-Policy reaching 0.8837 and 0.8698, respectively, higher compared with conventional VLAs as shown in Fig. 7. A possible explanation is the difference in training objectives: conventional VLAs primarily learn action imitation from successful trajectories, whereas world-model-based policies explicitly model action-conditioned state transitions, encouraging the model to reason about how actions affect the current state and future outcomes. Such a transition-aware representation may provide a stronger inductive bias for adapting to unexpected failure states, leading to more consistent recoverv behavior.

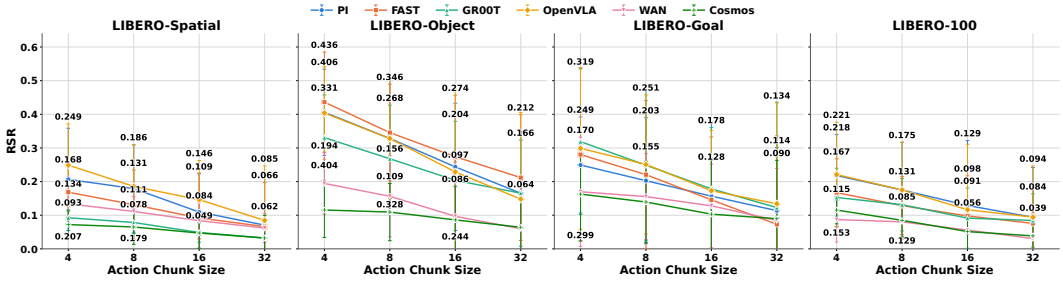


Figure 8: The impact of Chunk Size on Model Recovery Capability.

4.2.3 How to enhance model failure handling?

Key Finding 5: Smaller action chunks improve recovery by enabling more frequent state feedback. Across all four LIBERO subsets, recovery success consistently degrades as the action chunk size increases from 4 to 32, with the most substantial drop generally occurring at larger chunk sizes. This trend is shared across different policies, indicating that recovery performance is highly sensitive to the temporal granularity of action execution. Longer action chunks commit the robot to a sequence of actions before the next policy update, making it harder to correct deviations caused by failures. In contrast, smaller chunks enable more frequent feedback and action adjustment, allowing the policy to better track the evolving post-failure state. These results suggest that effective failure recovery requires not only a capable policy, but also sufficiently fine-grained closed-loop control.

Key Finding 6: Recovery training does not effectively transfer to failures encountered during standard task execution. Standard LIBERO evaluation is not failure free: even when starting from valid initial states, policies can err during execution and fail to complete the task. This raises a natural question: can training on failure-recovery scenarios teach a policy to recover from failures that arise during standard task execution, thereby improving its overall LIBERO success rate? To examine this, we jointly train the policies on the original LIBERO training data and LIBERO-Recover data, and evaluate them on both standard LIBERO tasks and LIBERO-Recover scenarios.

Table 3: Impact of temporal prompts on model recovery capability.

	LIBERO-Recover			
Method	LIBERO-Spatial	LIBERO-Object	LIBERO-Goal	LIBERO-100
GR00T N1.5	0.057	0.260	0.263	0.133
GR00T N1.5 (temporal)	0.066	0.260	0.264	0.166
OpenVLA-OFT	0.156	0.326	0.246	0.193
OpenVLA-OFT (temporal)	0.193	0.333	0.300	0.247

Interestingly, the results do not support this expectation. Adding LIBERO-Recover data consistently improves performance on LIBERO-Recover, but produces little or even negative change on standard LIBERO. For GR00T N1.5, mixed training improves the average recovery success rate from 17.8% to 21.2%, while the average LIBERO success rate only increases from 86.5% to 87.2%. For OpenVLA-OFT, recovery success increases from 20.8% to 25.4%, whereas the average LIBERO success rate slightly decreases from 97.1% to 96.6%. Thus, the recovery data clearly provides useful supervision for the recovery scenarios on which it is evaluated, yet this learned recovery capability provides little benefit when failures arise naturally during standard LIBERO execution.

This result reveals a failure-recovery transfer gap. Exposing a policy to failure states is not sufficient to make it reliably recover from its own execution errors during ordinary task execution. ***This***

suggests that effective recovery may depend not only on learning how to act from a failed state, but also on recognizing and responding to failures that emerge online during the policy's own execution trajectory.

Key Finding 7: Temporal context improves failure recovery. Providing the task initial frame as temporal context consistently improves recovery performance across both models, with the largest gains on more challenging tasks. For GR00T N1.5, temporal context improves the overall recovery success rate from 17.8% to 18.9%. The effect is substantially stronger for OpenVLA-OFT, whose average recovery success increases from 20.8% to 26.8%, including a notable improvement on LIBERO-100 from 19.3% to 24.7% and on LIBERO-Goal from 24.6% to 30.0%. These improvements suggest that recovery benefits from access to the task's temporal context rather than relying solely on the current failure state. The initial frame provides a reference for comparing the current scene against the intended task configuration, allowing the policy to identify state deviations and infer what has been disrupted by the failure. This indicates that effective failure recovery requires not only understanding the current state, but also reasoning about how the current state differs from the expected state.

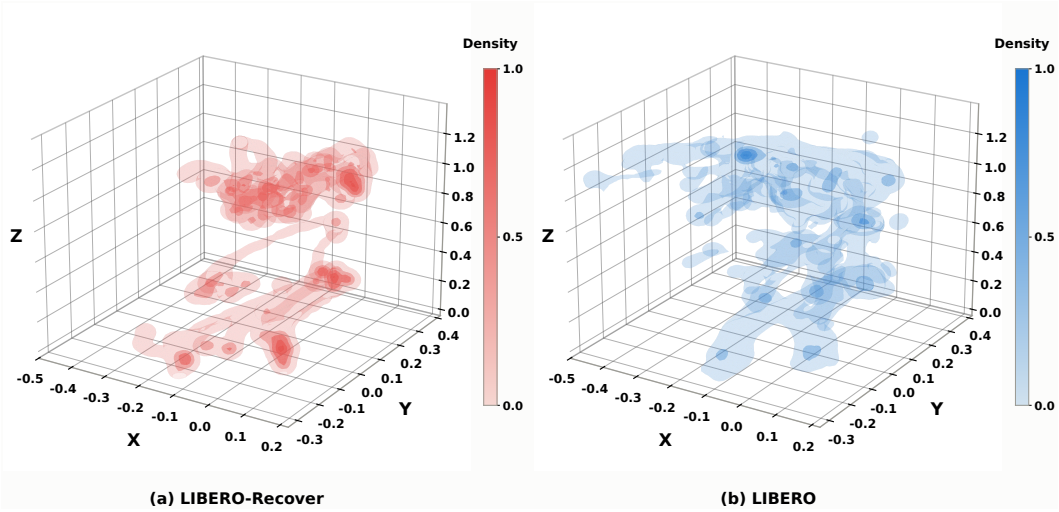


Figure 9: Distribution visualization of training trajectories on LIBERO-Recover and LIBERO.

Key Finding 8: Failure recovery occurs in a more concentrated state distribution. The KDE visualization reveals that LIBERO-Recover exhibits a more concentrated state distribution than the original LIBERO data, with several distinct high-density regions, while the original data covers a more dispersed workspace. This suggests that failure recovery is concentrated around a limited number of critical states where failures are more likely to occur, such as grasping, placing, and object interaction. In contrast, standard demonstrations cover a broader range of successful execution states. This distributional difference indicates that recovery data provides concentrated information around critical state-transition regions that are underrepresented in standard demonstrations, highlighting the need to explicitly model and evaluate failure recovery.

5 Conclusion

We introduced **LIBERO-Recover**, a benchmark for failure recovery in robotic manipulation. Using naturally occurring failures from embodied model executions, LIBERO-Recover evaluates whether models can understand disrupted states, adapt their actions, and recover to complete the original task. Experiments across VLA and WAM models reveal a substantial gap between standard task success and failure recovery, with performance degrading sharply as recovery requires deeper state reasoning. Our analysis shows that finer-grained action execution and temporal context can improve recovery, while recovery training provides limited transfer to failures during standard task execution. We hope LIBERO-Recover will encourage research toward robust and failure-aware embodied agents.

AI USE STATEMENT

During the preparation of this work, the authors used generative AI tools to assist with method code implementation, language polishing, and figure layout and color adjustments. Specifically, the open-source Qwen 3.5 model was applied to suggest the partitioning of the V_{pre}, V_{fail}, and V_{post} sets, and all model-generated annotations were manually sampled and cross-verified by the authors to guarantee data quality. Generative AI was not used to develop theoretical models or conceptual frameworks, or to formulate or refine hypotheses. It was not used to provide essential elements for mathematical proofs, assist in writing proofs, or support qualitative or thematic data analysis.

All AI-assisted outputs were thoroughly reviewed and verified by the authors. Any code or annotations generated with the assistance of large language models were independently tested and verified for correctness. The authors take full responsibility for the final content of this work, including all text, claims, data annotations, and materials produced with the assistance of generative AI. Please refer to the Appendix for further details.

REPRODUCIBILITY STATEMENT

Section 2 and Section 3 details the methodology for constructing LIBERO-Recover and the data collection procedure. Subsequent experimental section 4 and the Appendices B and E describe the training datasets, benchmark splits, evaluation metrics, and baseline configurations. The Appendix further provides full model configurations, optimization hyperparameters, evaluator definitions, and metric rules. Upon acceptance, we will publicly release all source code, datasets, evaluation protocols, and model checkpoints required to reproduce our findings.

REFERENCES

Hassan Abu Alhaija, Jose Alvarez, Maciej Bala, Tiffany Cai, Tianshi Cao, Liz Cha, Joshua Chen, Mike Chen, Francesco Ferroni, Sanja Fidler, et al. Cosmos-transfer1: Conditional world generation with adaptive multimodal control. *arXiv preprint arXiv:2503.14492*, 2025.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. Univla: Learning to act anywhere with task-centric latent actions. *arXiv preprint arXiv:2505.06111*, 2025.

Remi Cadene, Simon Alibert, Francesco Capuano, Michel Aractingi, Adil Zouitine, Pepijn Kooijmans, Jade Choghari, Martino Russi, Caroline Pascal, Steven Palma, et al. Lerobot: An open-source library for end-to-end robot learning. In *International Conference on Learning Representations*, volume 2026, pp. 122398–122417, 2026.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

StarVLA Community. Starvla: A lego-like codebase for vision-language-action model developing. *arXiv preprint arXiv:2604.05014*, 2026.

Rongxu Cui, Zongzheng Zhang, Jingrui Pang, Haohan Chi, Jinbang Guo, Saining Zhang, Shaoxuan Xie, Xin Jin, Yao Mu, Jiaolong Yang, et al. Libero-safety: A comprehensive benchmark for physical and semantic safety in vision-language-action models. *arXiv preprint arXiv:2606.23686*, 2026.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, et al. Libero-plus: In-depth robustness analysis of vision-language-action models. *arXiv preprint arXiv:2510.13626*, 2025.

Yucheng Hu, Yanjiang Guo, Pengchao Wang, Xiaoyu Chen, Yen-Jen Wang, Jianke Zhang, Koushil Sreenath, Chaochao Lu, and Jianyu Chen. Video prediction policy: A generalist robot policy with predictive visual representations. *arXiv preprint arXiv:2412.14803*, 2024.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

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.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, et al. Cosmos policy: Fine-tuning video models for visuomotor control and planning. *arXiv preprint arXiv:2601.16163*, 2026.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Jiaming Liu, Mengzhen Liu, Zhenyu Wang, Pengju An, Xiaoqi Li, Kaichen Zhou, Senqiao Yang, Renrui Zhang, Yandong Guo, and Shanghang Zhang. Robomamba: Efficient vision-language-action model for robotic reasoning and manipulation. *Advances in Neural Information Processing Systems*, 37:40085–40110, 2024.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Wenxuan Song, Ziyang Zhou, Han Zhao, Jiayi Chen, Pengxiang Ding, Haodong Yan, Yuxin Huang, Feilong Tang, Donglin Wang, and Haoang Li. Reconvla: Reconstructive vision-language-action model as effective robot perceiver. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 18549–18557, 2026.

Shiang-Feng Tsai, Jin-Cheng Jhang, Yen-Ling Tai, Jia-Hong Lai, Shih-Yun Wong, Yi-Ting Chen, et al. Direct action-head injection of a grounded 3d point unlocks spatial and task generalization. *arXiv preprint arXiv:2606.27663*, 2026.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Guodong Wang, Chenkai Zhang, Qingjie Liu, Jinjin Zhang, Jiancheng Cai, Junjie Liu, and Xinmin Liu. Libero-x: Robustness litmus for vision-language-action models. *arXiv preprint arXiv:2602.06556*, 2026.

Ruijie Zheng, Jing Wang, Scott Reed, Johan Bjorck, Yu Fang, Fengyuan Hu, Joel Jang, Kaushil Kundalia, Zongyu Lin, Loic Magne, et al. Flare: Robot learning with implicit world modeling. *arXiv preprint arXiv:2505.15659*, 2025.

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. Libero-pro: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2025.

Chuning Zhu, Raymond Yu, Siyuan Feng, Benjamin Burchfiel, Paarth Shah, and Abhishek Gupta. Unified world models: Coupling video and action diffusion for pretraining on large robotic datasets. *arXiv preprint arXiv:2504.02792*, 2025.

APPENDIX CONTENTS

A Related Work **15**

A.1 Robot Manipulation Policies 15

A.2 Robot Manipulation Benchmarks 15

B Dataset Construction and Annotation **16**

B.1 Operational Definition of Failure Stages 16

B.2 Prompt for Temporal Failure Localization 16

B.3 Prompt for Recovery Difficulty and Failure Categorization 17

B.4 Teleoperation Data Collection 19

B.5 Trajectory Processing 19

C Dataset and Release Format **20**

C.1 Evaluation Recovery Scenarios 20

C.1.1 BDDL Format 21

C.1.2 BDDL Parsing Constraint. 22

C.2 Expert Recovery Trajectories 22

C.2.1 Data Format 22

C.2.2 Dataset Scale and Coverage 23

C.3 Pre-failure Temporal Histories 23

D Recovery Evaluation Protocol **23**

D.1 Evaluation Pipeline 23

D.2 Failure-State Perturbation 24

D.3 Gripper Initialization 25

D.4 Recovery Rollout 26

D.5 Metric Aggregation 27

E Experimental Implementation Details **29**

E.1 Model Adaptation 29

E.2 Unified Training of the Evaluated Policies 30

E.3 Recovery-Aware Fine-Tuning Configuration 30

F Failure Analysis **32**

F.1 Failure to Verify Action Outcomes 32

F.2 Insufficient Fine-Grained Control 32

F.3 Recovery Requires an Intermediate Action 33

F.4 Recovery Degradation under Repeated Failures 34

G Challenges in Failure Recovery **35**

G.1 Scene and Robot State Changes 36
G.2 Grasping under Different Object Configurations 36
G.3 Implications for Recovery Performance 36

H Limitations and Discussion **37**

A Related Work

A.1 Robot Manipulation Policies

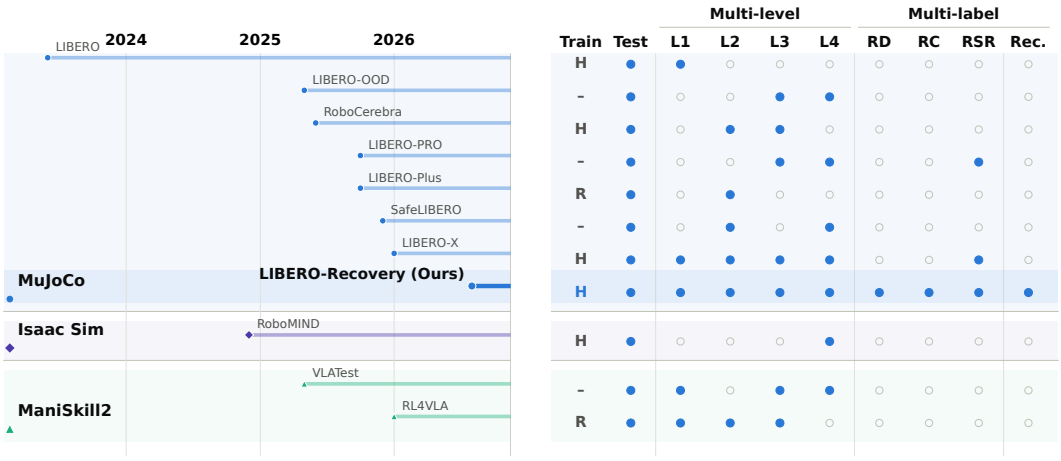


Figure 10: **Comparison of LIBERO-Recover with existing benchmarks.** LIBERO-Recover is the first benchmark dedicated to failure recovery, where **Rec.**

Robot manipulation policies have evolved from visual imitation learning toward general-purpose models that jointly understand vision, language, and actions. Representative VLA methods, including RT-1, RT-2, OpenVLA Kim et al. (2024), π_0 Black et al. (2024), $\pi_{0.5}$ Intelligence et al. (2025), RoboMamba Liu et al. (2024), and GR00T Bjorck et al. (2025), leverage large-scale vision-language and robot datasets to improve generalization across tasks, objects, and embodiments.

More recently, world models have been incorporated into robot manipulation, leading to WAMs such as UniVLA Bu et al. (2025), Video Policy Hu et al. (2024), UWM Zhu et al. (2025), FLARE Zheng et al. (2025), and Cosmos-Policy Kim et al. (2026). By modeling actions and future states, these methods provide foresight into action consequences and improve long-horizon planning. However, existing VLA/WAM methods primarily focus on successful action prediction and execution, while systematic modeling of failures, unexpected state transitions, and post-failure recovery remains largely unexplored. Enabling robots to understand, adapt to, and recover from failures is therefore crucial for reliable manipulation.

A.2 Robot Manipulation Benchmarks

Standardized benchmarks are essential for evaluating the generalization and robustness of robot manipulation policies. Early benchmarks such as RLBench James et al. (2020) and LIBERO Liu et al. (2023) established large-scale simulation tasks and unified evaluation protocols. Recent benchmarks, including CALVIN Mees et al. (2022), SimplerEnv Li et al. (2024), RoboCasa Nasiriany et al. (2024), RoboTwin Chen et al. (2025), and LIBERO-Plus Fei et al. (2025), LIBERO-Pro Zhou et al. (2025), LIBERO-X Wang et al. (2026), and LIBERO-Safety Cui et al. (2026), further evaluate task generalization, scene diversity, long-horizon interaction, and robustness to controlled changes in object placement, appearance, robot pose, and task instructions. Despite these advances, existing benchmarks primarily rely on *manually predefined distribution shifts* and evaluate whether policies can *complete tasks under changed conditions, as shown in Fig. 10*.

In contrast, real execution failures can induce complex and unstructured states, such as object displacement and unintended state changes, which remain largely unexplored. **LIBERO-Recover** addresses this gap by collecting naturally occurring failures and resulting state transitions from embodied model executions, constructing progressively challenging recovery scenarios to evaluate state understanding, action adaptation, interaction reasoning, and failure recovery.

B Dataset Construction and Annotation

B.1 Operational Definition of Failure Stages

For temporal analysis, each execution video is divided into three phases: Pre-Failure, During-Failure, and Post-Failure.

Pre-Failure. The Pre-Failure phase contains the observations before the first task-relevant deviation. During this phase, the robot follows an execution that is still consistent with the task instruction, and the current plan remains executable.

During-Failure. The During-Failure phase starts at the earliest observable moment at which the robot action causes a task-relevant deviation. It includes the causal interaction that produces the failure and ends when the resulting state becomes visually stable or the robot stops attempting the original action.

Post-Failure. The Post-Failure phase begins after the failure state has become stable. It contains the observations available after the failure and before recovery, termination, or manual intervention. These observations characterize the state that a recovery policy must interpret before generating corrective actions.

The failure timestamp corresponds to the beginning of the During-Failure phase. All timestamps are measured from the beginning of the recorded video and rounded to one decimal place.

B.2 Prompt for Temporal Failure Localization

We use a vision-language model to localize the onset of execution failures from recorded robot videos. The following prompt was used as the default template. The original prompt was translated into English for clarity in the paper; the implementation used the corresponding Chinese version.

Temporal Failure Localization Prompt

```
You are an expert in analyzing robot manipulation videos.
Watch the entire video carefully and identify the earliest moment at which
the robot experiences a task-relevant execution failure.
A failure is defined as the earliest observable moment when:
   (1) the robot fails to grasp or place the manipulated object as intended;
   (2) the robot action causes an object displacement or interaction that
invalidates the current execution plan.
Return the earliest failure timestamp for the manipulated object.
Requirements:
   1. Analyze the complete video before making a decision.
   2. Report time in seconds from the beginning of the video.
   3. Round the timestamp to one decimal place.
   4. Do not provide explanations.
   5. Output JSON only.
The output format must be exactly:
    {
      "fail": 2.0
    }
```

The model output is parsed as a JSON object. If the generated response contains additional text, the content after the final reasoning marker is first extracted; a regular-expression fallback is then used to recover the first valid JSON object. Samples that cannot be parsed into the required schema are discarded and reviewed separately.

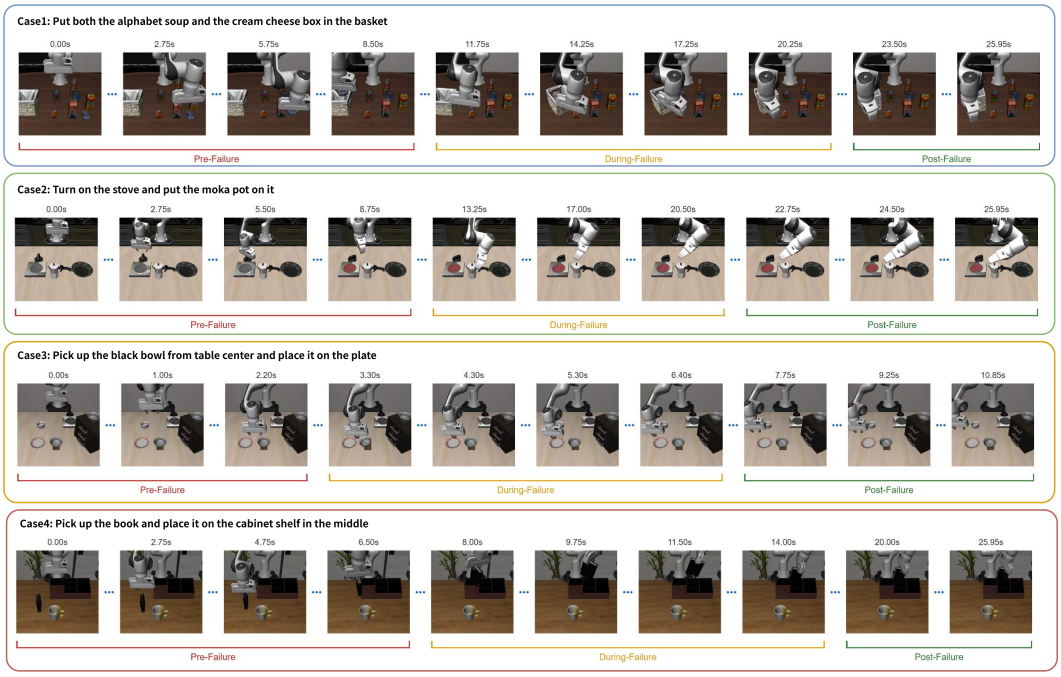


Figure 11: Qualitative examples of temporal failure localization. Each row shows ten representative frames from a manipulation rollout, with timestamps from the original video. The brackets below the frames indicate Pre-Failure, During-Failure, and Post-Failure, illustrating the progression from task execution to failure development and its resulting state. Frames are sampled non-uniformly to highlight relevant transitions.

Qualitative Examples. Figure 11 illustrates temporal failure localization across four manipulation tasks: placing two objects in a basket, placing a moka pot on an activated stove, transferring a bowl onto a plate, and placing a book on a cabinet shelf. Each row presents ten representative frames in chronological order, with timestamps indicating their positions in the original video. The phase annotations distinguish the execution preceding the failure, the interaction during which the deviation develops, and the resulting post-failure configuration.

Temporal Context Evidence. These examples highlight the temporal evidence needed to interpret an execution failure. Changes in the gripper–object relationship, object position, and task progress help distinguish an ongoing manipulation attempt from an observable failure. The localization prompt returns the failure onset, corresponding to the transition from Pre-Failure to During-Failure; the three-phase annotations provide additional context for visualizing the complete failure process.

B.3 Prompt for Recovery Difficulty and Failure Categorization

After localizing the failure timestamp, we further annotate each failure with three complementary fields: the recovery difficulty level, the primary failure category, and the minimum corrective action required to resume the original task. The recovery action is expressed as a concise executable instruction rather than a causal explanation. This additional field makes the annotation more operational and supports subsequent human verification by teleoperators.

The following prompt is used as the default template.

title

```
You are an expert in robot manipulation and failure recovery.
Watch the entire video carefully and classify the observed failure according to
its recovery difficulty and primary failure category. Then describe the minimum
```

corrective action that the robot should execute to recover from the failure and resume the original task.

Use the following recovery levels:

L1 -- Action Retry: the failure can be corrected by repeating essentially the same action, without any meaningful change to the object or environment.

L2 -- Action Adaptation: the robot must modify the execution of the action, such as changing the grasp position, approach direction, timing, or motion trajectory.

L3 -- Object State Recovery: the failure changes the state of an object, and the robot must first restore or rearrange the object before continuing the original task.

L4 -- Environmental Recovery: the failure changes the surrounding environment or creates an obstruction that must be removed or restored before the task can continue.

Choose exactly one failure category from the following list:

grasp_failure

placement_failure

object_displacement

collision_or_interaction_failure

environmental_obstruction

other

Requirements:

1. Use the original task instruction as the task objective.

2. Consider the minimum corrective behavior required for successful recovery.

3. Assign exactly one recovery level and one failure category.

4. Describe the recovery action as a concise imperative instruction.

5. The recovery action should specify the relevant object, target, gripper state, spatial adjustment, and execution order when necessary.

6. Do not provide explanations, causes, confidence scores, or alternative recovery plans.

7. Output valid JSON only. Do not use Markdown code fences.

The output format must be exactly:

{ "recovery_level": "L2", "failure_category": "grasp_failure", "recovery_action": "Reposition the gripper above the bowl, close the gripper to grasp it, and resume the original task." }

The resulting JSON annotation contains both a semantic description of the failure and an executable recovery specification. For example, an action retry may be represented by repeating the failed grasp, whereas an object-state failure may require first repositioning the displaced object and then resuming the original instruction.

The generated JSON annotations are subsequently reviewed by four teleoperators with experience in robot manipulation. Each reviewer is provided with the complete video, the original task instruction, and the model-generated JSON output. The reviewers verify three aspects independently: whether the recovery level matches the minimum corrective behavior, whether the failure category describes the primary failure mechanism, and whether the proposed recovery action is physically executable and sufficient to resume the original task. Reviewers may accept the annotation, revise one or more fields, or flag the sample as ambiguous. Samples with reviewer disagreement are forwarded for adjudication, and the final annotation records the reviewed recovery level, failure category, and recovery action.

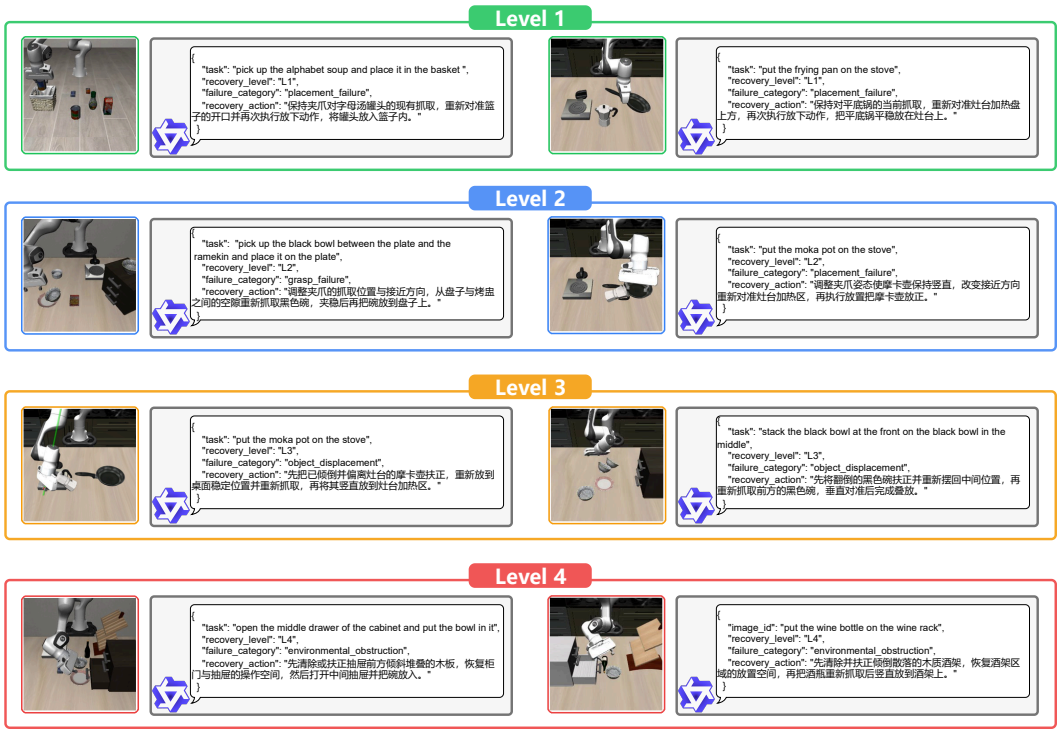


Figure 12: Examples of model-generated recovery annotations. Each example shows the observed failure, the predicted recovery level and failure category, and the corresponding corrective action generated in the JSON output.

B.4 Teleoperation Data Collection

For each selected failure scenario, the simulator is restored to the recorded failure state while preserving the original task instruction and scene configuration. A human teleoperator is then asked to recover the task from this state using a SpaceMouse.For each scenario, multiple trajectories may be recorded when different valid recovery behaviors are possible.

Before each rollout, the robot pose, gripper state, object states, and environment state are reset to the scenario-specific configuration. The teleoperator is not provided with the original successful action sequence and must determine the recovery behavior from the current observation and the task instruction. A recovery trajectory is considered successful if the original task goal is satisfied after the failure state.

B.5 Trajectory Processing

The teleoperated demonstrations are organized in the episode-based LeRobot v2.1 format Cadene et al. (2026). Each recovery episode contains synchronized third-person and wrist-camera observations, robot states, and end-effector actions generated during teleoperation. In addition to the robot observations and actions, standard temporal and indexing fields, such as timestamps, frame indices, episode indices, and task indices, are used to preserve temporal ordering and task correspondence within each trajectory.

During policy training, the LeRobot data pipeline reconstructs temporally aligned samples by jointly indexing visual observations, robot states, task information, and actions, making the demonstrations directly compatible with LeRobot-based VLA training pipelines such as StarVLA Community (2026). Importantly, the expert recovery trajectories are released as training and adaptation data rather than as part of the benchmark input. Standard LIBERO-Recover evaluation requires only the recovery-scene assets, and therefore does not provide the expert recovery action sequence to the evaluated policy.

Table 4: Source task suites and BDDL directories used by LIBERO-Recover, covering 130 tasks in total. LIBERO-100 comprises `libero_90` (90 tasks) and `libero_10` (10 tasks). Original task instructions and scene definitions are preserved, while recovery episodes are initialized from recorded failure states.

Suite	bddl folder	Subtasks	Scope
LIBERO-Spatial	`libero_spatial`	10	One bowl, ten spatial relations (between / next to / on / inside)
LIBERO-Object	`libero_object`	10	Ten distinct objects, each placed into the basket
LIBERO-Goal	`libero_goal`	10	One scene, ten distinct goal states (open / close / on / turn on)
LIBERO-100	`libero_90` + `libero_10`	100	90 single-family long-horizon tasks (cabinets, stoves, baskets, trays, caddies) and 10 compositional multi-step tasks
Total		**130**	

C Dataset and Release Format

LIBERO-Recover is released as a collection of recovery-oriented scene assets and optional training datasets. The release preserves the task semantics and scene descriptions of LIBERO, while additionally recording the simulator state at selected timestamps during trajectory execution. This design separates the task definition from the physical state used to initialize a recovery episode.

LIBERO-Recover is organized into three complementary data components: *Evaluation Recovery Scenarios*, *Expert Recovery Trajectories*, and *Pre-failure Temporal Histories*. They support different stages of the recovery pipeline, including scene-based evaluation, recovery policy training, and temporal-context modeling.

Source Task Suites. Table 4 summarizes the source task suites and their corresponding BDDL directories. LIBERO-Recover covers 130 tasks across four suites: LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-100. The latter comprises LIBERO-90 and LIBERO-10, resulting in five directories in the released task assets.

C.1 Evaluation Recovery Scenarios

The *Evaluation Recovery Scenarios* provide the core assets for benchmark evaluation. Each scenario is associated with a LIBERO task and contains a BDDL task description. These scenarios are intended for standardized evaluation. They allow different policies to start from the same task and recorded scene state, while enabling multiple recovery trials through controlled variations in the initialization procedure.

Each recovery scene is represented by a pair of files: a BDDL task description and a corresponding simulator-state file:

$$\mathcal{S}_i = (\mathcal{B}_i,\, \mathcal{X}_i)\,,$$

where $\mathcal{B}_i$ is a BDDL file and $\mathcal{X}_i$ is a NumPy file storing the flattened MuJoCo simulation state. The filename stem is shared by the two files.

The scene assets are organized by task suite, task instance, and episode. A typical directory structure is shown below:

```
selected_bddl_scenes_<suite>/
+-- <task_name>/
    +-- episode_<id>/
        |-- step_<timestamp>.bddl
        +-- step_<timestamp>_sim_state.npy
```

For example, `step_0400.bddl` and `step_0400_sim_state.npy` jointly describe the scene state recorded at a particular execution timestamp.

C.1.1 BDDL FORMAT

The Evaluation Recovery Scenarios use the BDDL format inherited from LIBERO. Each BDDL file defines the symbolic task and the corresponding scene layout, while the physical simulator state at the recorded timestamp is stored separately in a paired `_sim_state.npy` file. A typical BDDL file follows the template below:

Simplified BDDL Template

```
(define (problem <problem_name>)
  (:domain robosuite)
  (:language <task_instruction>)

  (:regions
    (<object_init_region>
      (:target <support_surface>)
      (:ranges ((x_min y_min x_max y_max)))
      (:yaw_rotation ((yaw_min yaw_max)))
    )
    ...
    (<task_region>
      (:target <target_fixture>)
    )
  )

  (:fixtures
    <fixture_1> - <fixture_type>
    ...
  )

  (:objects
    <object_1> - <object_type>
    ...
  )

  (:obj_of_interest
    <object_1>
    ...
  )

  (:init
    (On <object_1> <object_1_init_region>)
    ...
  )

  (:goal
    (And
      <goal_predicate_1>
      ...
    )
  )
)
```

The `:language` field specifies the natural-language task instruction. The `:fixtures` and `:objects` fields define the static scene elements and movable objects. The `:regions` field specifies object initialization regions and task-specific target regions. The `:init` block associates objects with their corresponding initialization regions, whereas the `:goal` block preserves the original symbolic task objective.

C.1.2 BDDL PARSING CONSTRAINT.

During dataset preparation, we observed that the BDDL parser may fail when objects of the same type are declared on separate lines. For example, in a scene containing two plates, the following declaration can trigger a parsing error:

```
(:objects
  plate_1 - plate
  plate_2 - plate
)
```

When this error occurs, the declarations must be rewritten as a single typed list:

```
(:objects
  plate_1 plate_2 - plate
)
```

All objects sharing the same type should therefore be declared on one line in the released BDDL files. This requirement is imposed by the specific BDDL parsing implementation used in our LIBERO-Recover pipeline. It only affects the textual representation of the task definition and does not change the identities, properties, or semantics of the objects.

C.2 EXPERT RECOVERY TRAJECTORIES

The *Expert Recovery Trajectories* provide human demonstrations of successful recovery behaviors. Each trajectory records robot observations, robot states, actions, and task-related metadata throughout the recovery process. These trajectories capture how human experts respond to disrupted scenes, correct states caused by execution failures, and continue execution toward the original task goal.

The demonstrations can be used for supervised policy training, recovery policy adaptation, or qualitative analysis of recovery behaviors. In contrast to the evaluation scenarios, which define the initial conditions for benchmarking, the expert trajectories provide action-level supervision from failure states, enabling models to learn how recovery should be executed.

C.2.1 DATA FORMAT

The recovery demonstrations are released in the LeRobot format Cadene et al. (2026), which provides a standardized representation for multimodal robot-learning trajectories. Following its episode-based organization, low-dimensional state–action data and visual observations are stored separately while remaining temporally synchronized within each recovery episode.

The low-dimensional trajectory data are stored in Parquet files and contain robot states, end-effector actions, and temporal indices. Visual observations are encoded separately as video streams, including synchronized third-person and wrist-camera views. This separation follows the LeRobot design principle of storing high-frequency tabular robot signals and visual observations using storage formats suited to their respective modalities Cadene et al. (2026).

At the dataset level, the release is organized into three major components: `data`, `videos`, and `meta`. The `data` component stores state–action trajectories, `videos` stores the corresponding camera observations, and `meta` maintains dataset-, task-, and episode-level information. A representative organization is shown below:

```
LIBERO_Recovery_Expert/
|-- data/
|   `-- chunk-xxx/
|       |-- episode_xxxxxx.parquet
|       `-- ...
|-- videos/
|   `-- chunk-xxx/
|       |-- <third-person-camera>/
|       |   `-- episode_xxxxxx.mp4
```

```
|       `-- <wrist-camera>/
|           `-- episode_xxxxxx.mp4
`-- meta/
    |-- info.json
    |-- episodes.jsonl
    |-- episodes_stats.jsonl
    `-- tasks.jsonl
```

This standardized representation facilitates integration with existing LeRobot-based robot-learning pipelines and VLA training frameworks such as StarVLA Community (2026). It also makes the recovery demonstrations compatible with the broader LIBERO Liu et al. (2023) data ecosystem, enabling recovery data to be combined with conventional task demonstrations during policy adaptation.

C.2.2 Dataset Scale and Coverage

The released dataset contains 3,184 human recovery episodes collected by four teleoperators using a SpaceMouse from 413 failure scenarios. In total, the dataset contains 625,731 trajectory frames and covers 110 task types.

All trajectories are recorded at 20 Hz and provide two synchronized visual streams, including a third-person view and a wrist-camera view, together with robot joint states and end-effector actions. Consequently, the dataset preserves the complete observation–state–action evolution of the recovery process rather than only the final recovery outcome.

In contrast to the original LIBERO demonstrations Liu et al. (2023), which primarily capture task execution from standard task initializations, the recovery trajectories start from disrupted states and explicitly capture the sequence of actions required to restore progress toward the original task objective.

C.3 Pre-failure Temporal Histories

The *Pre-failure Temporal Histories* contain short temporal segments extracted from LIBERO-10 execution trajectories prior to failure. We use Qwen3.5-27B-Instruct to analyze the complete execution videos and temporally localize the point at which the rollout begins to deviate from the expected task progression. The resulting timestamp is aligned with the original trajectory, from which a preceding window of observations and robot states is extracted as the pre-failure history. These sequences preserve the dynamic context of how the robot approaches an abnormal state, including changes in object poses, end-effector motion, and gripper state.

The pre-failure histories are provided as an optional temporal-context component. A policy may use only the current recovery observation, or additionally access the preceding history when temporal memory is supported.

D Recovery Evaluation Protocol

D.1 Evaluation Pipeline

LIBERO-Recover evaluates whether a manipulation policy can resume an original task after execution has entered a disrupted state. Figure 13 summarizes the complete evaluation protocol, which consists of scenario initialization, policy–environment interaction, recovery rollout, and metric aggregation. Unlike conventional LIBERO evaluation, where a policy starts from a nominal task initialization, each recovery episode starts from a simulator state associated with a recorded execution failure. The original task instruction and goal condition are preserved, while the physical state provided to the policy reflects the failure context and the controlled initialization protocol.

For a policy π and a recovery scenario $\mathcal{S}_i$, the evaluation procedure constructs an initialized state

$$\widetilde{s}_i = \mathcal{I}\left(s_i^{\mathrm{fail}}, \delta_i^{\mathrm{obj}}, g_i^{\mathrm{init}}\right), \tag{13}$$

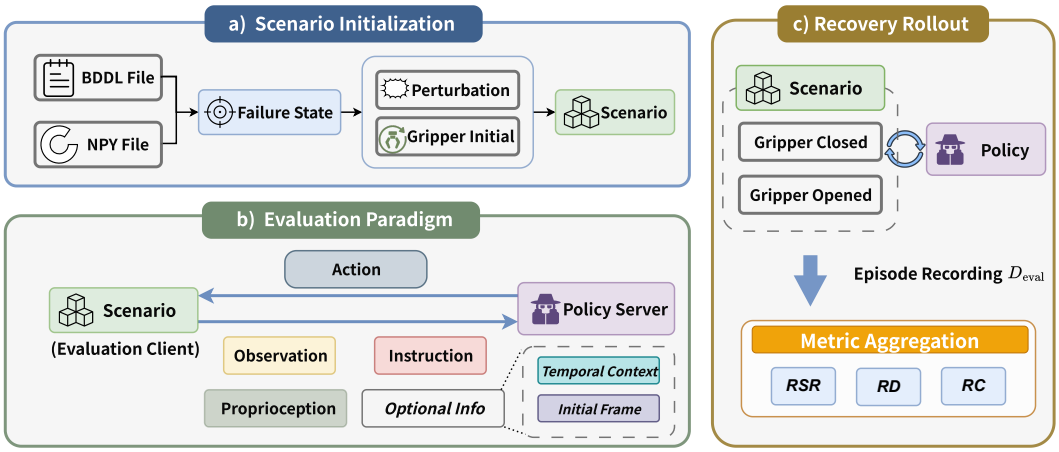


Figure 13: **Overview of the LIBERO-Recover evaluation pipeline.** The evaluation consists of three stages. (a) *Scenario Initialization*: the BDDL task definition and the recorded simulator state are used to reconstruct a failure state, after which object perturbation and gripper initialization produce an initialized recovery scenario. (b) *Evaluation Paradigm*: the evaluation client sends the current observation, task instruction, robot proprioception, and optional temporal context to the policy server, which returns the next action. (c) *Recovery Rollout*: the policy is evaluated under open- and closed-gripper conditions, producing episode-level evaluation records that are aggregated into RSR, RD, and RC.

where s_i^{fail} denotes the recorded failure state, δ_i^{obj} denotes the object-state perturbation, and g_i^{init} denotes the gripper initialization condition.

The policy then receives the original task instruction together with the observation generated from $\widetilde{s}_i$ and produces a closed-loop action sequence

$$\tau_i^{\text{rec}} = (\widetilde{s}_i, a_0, s_1, a_1, \ldots, s_T)\,. \tag{14}$$

The episode outcome is determined by whether the resulting state satisfies the original task goal within the allowed interaction horizon.

The complete evaluation process contains four logically ordered stages. First, the recorded failure state is restored to reconstruct the physical configuration from which recovery begins. Second, controlled initialization is applied to generate the evaluation condition, including object-state perturbation and gripper-state initialization. These operations modify the physical starting condition while preserving the task instruction and symbolic goal. Third, the policy interacts with the environment in a closed-loop manner, repeatedly using the current observation to generate actions until the task is completed or the rollout terminates. Finally, the complete trajectory, initialization condition, and task outcome are recorded as episode-level evaluation evidence and aggregated across episodes, tasks, recovery levels, and task suites.

Our implementation uses the unified visual–language–action interface of StarVLA Community (2026) to connect the policy with the LIBERO environment. The evaluation protocol itself is independent of a particular policy architecture: different policies are provided with the same task instructions, initialized states, perturbation conditions, and interaction budget. Consequently, the measured performance reflects the policy's ability to interpret and recover from disrupted task states rather than differences in episode construction.

D.2 Failure-State Perturbation

After restoring the recorded simulator state, we construct a recovery-oriented initial condition through two controlled interventions: object-state perturbation and gripper-state initialization. For each free-joint object, we independently sample a uniform displacement in the horizontal plane and apply the resulting offset to its position. In the default setting, the maximum displacement is $\pm 2\,\text{cm}$. This small perturbation is intended to assess local generalization around recorded failure states while

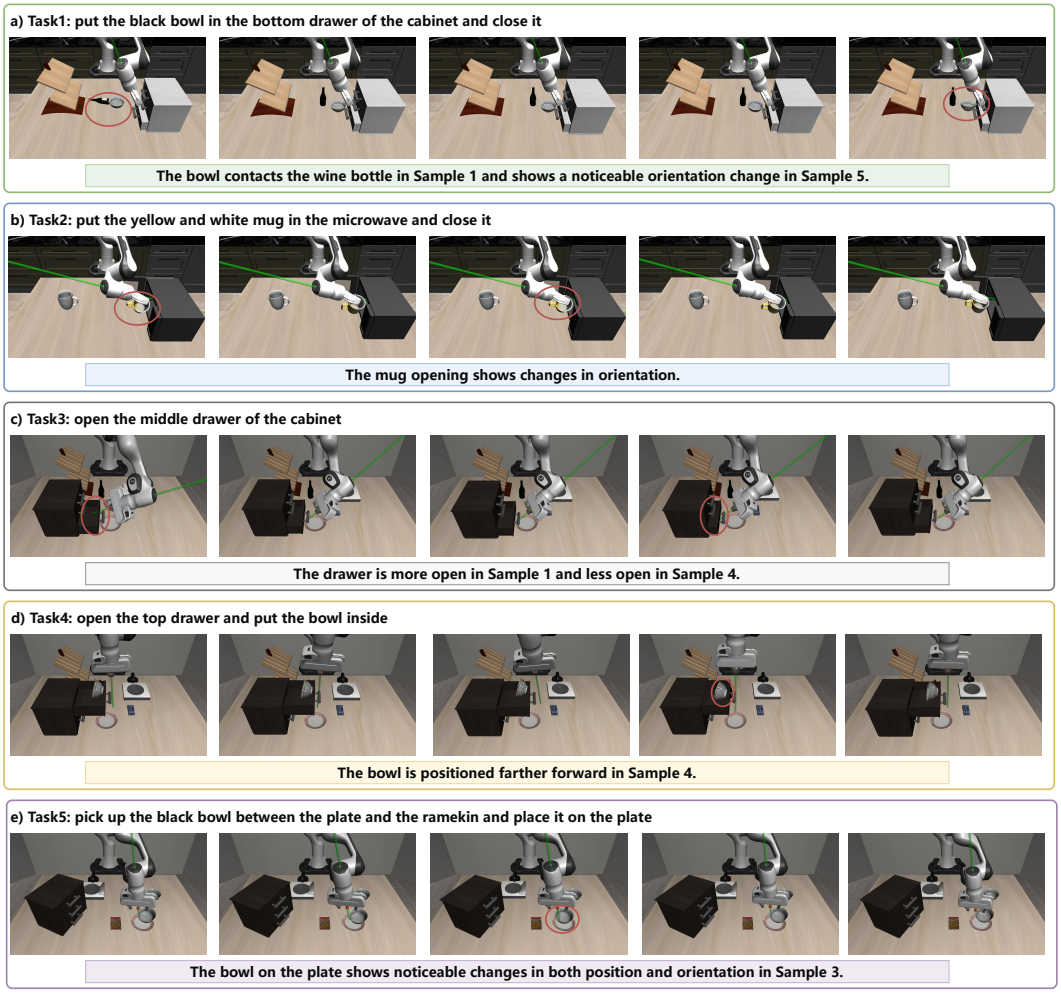


Figure 14: Controlled object perturbations across five representative tasks, with five samples per task. Small horizontal displacements (± 2 cm) introduce local variations while largely preserving the overall failure configurations shown here. This design probes local generalization around recorded failure states. Circled regions highlight changes in object position, contact, or orientation.

limiting changes to the underlying failure configuration. Objects controlled by hinge or slide joints, such as drawers and stove buttons, are excluded from this operation.

Figure 14 presents five perturbation samples for each of five representative tasks. Although all samples preserve the original task instruction and scene semantics, the perturbed objects exhibit different positions and orientations. For example, the bowl in the first sample is displaced sufficiently to contact the wine bottle, while the bowl in the fifth sample shows a noticeable orientation change. These examples illustrate how small, controlled state variations produce distinct recovery-oriented initial conditions.

D.3 Gripper Initialization

Following the state perturbation, the robot is initialized under one of two gripper conditions. In the open-gripper setting, the gripper is driven to its nominal open configuration. In the closed-gripper setting, the gripper starts fully closed, creating an abnormal initial condition that is typically not encountered in standard LIBERO execution. The target gripper configuration is reached by executing zero-delta arm actions for a fixed number of settling steps rather than by directly teleporting the joint state. The controller goal is also re-anchored to the current end-effector pose after the perturbation,

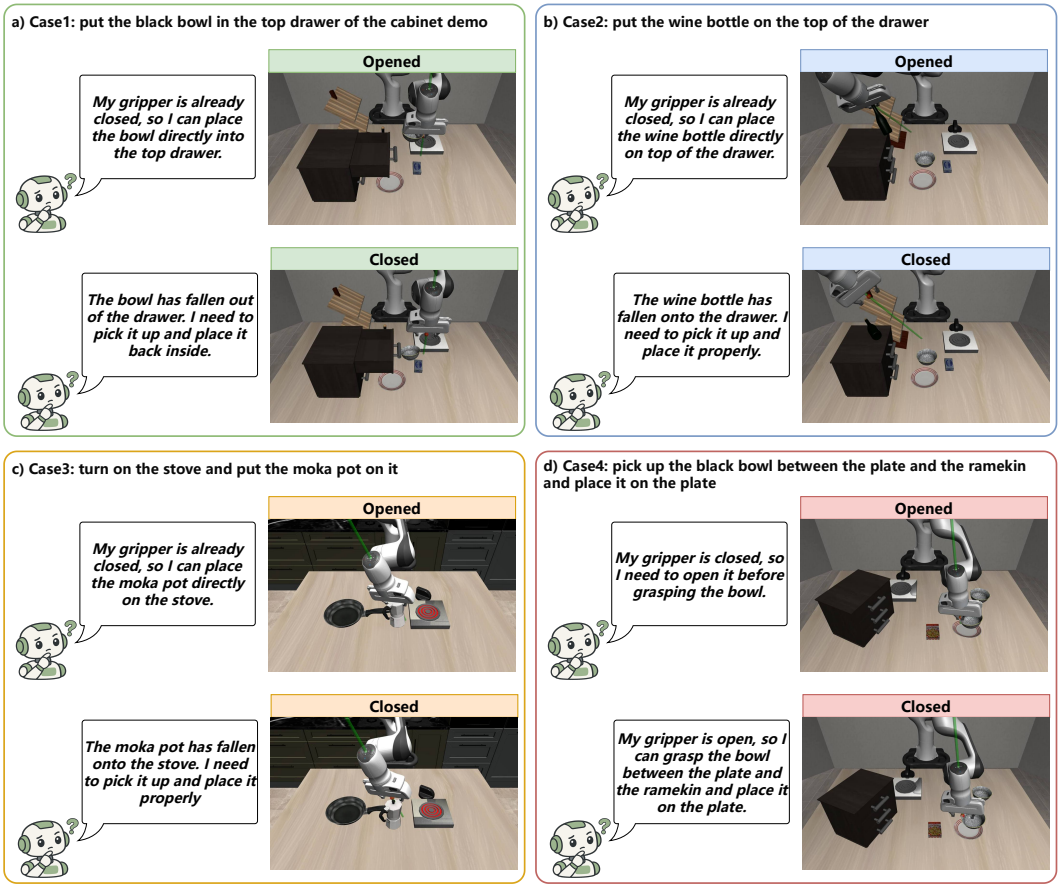


Figure 15: Examples of the two gripper initialization conditions used in recovery evaluation. The robot starts with either an open or a fully closed gripper, while the arm pose and surrounding scene remain unchanged. The two conditions require different responses from the policy, such as directly executing the task or first releasing the gripper to recover the intended manipulation state.

preventing an artificial arm displacement at the beginning of the rollout. Figure 15 shows the two initialization conditions used in the evaluation protocol.

For each scenario, the evaluator performs a fixed number of open-gripper and closed-gripper trials under the same task definition. The resulting evaluation splits isolate the effects of object-state perturbations and abnormal gripper initialization, while the fixed random seed ensures that the sampled perturbations can be reproduced across different policies.

The two gripper conditions may lead to different effective task states even when the arm pose and scene configuration are identical. With an open gripper, the policy can begin the intended manipulation directly, such as grasping an object, lifting it, or placing it at the target location. With a closed gripper, however, the robot may need to release the gripper before executing the nominal task, or respond to an object that has been displaced or dropped during initialization. Thus, gripper initialization does not simply change the actuator state; it creates distinct decision points that require the policy to interpret the current scene and select an appropriate recovery action. The examples in Figure 15 illustrate these task-dependent responses across different manipulation instructions.

D.4 Recovery Rollout

After the recovery scenario has been initialized, the policy starts from the resulting state $\widetilde{s}_0$ and attempts to complete the original task. At each control step t, the policy receives the original language instruction l, the current visual observation o_t, and the robot proprioceptive state r_t. When temporal

context is enabled, a history representation $\mathcal{H}_t$ is additionally provided. The policy then generates the next action according to

$$a_t = \pi_\theta \left(l, o_t, r_t, \mathcal{H}_t \right), \tag{15}$$

where $\mathcal{H}_t$ is omitted for policies evaluated without temporal context.

The predicted action is executed in the environment, resulting in a new physical state and the corresponding observation:

$$s_{t+1} = f_{\text{env}}(s_t, a_t), \qquad (o_{t+1}, r_{t+1}) = h(s_{t+1}). \tag{16}$$

The newly generated observation is then fed back to the policy for the next decision. This process forms a closed-loop recovery interaction in which the policy must continuously adapt its actions to the state produced by its own previous decisions. In contrast to replaying a fixed successful trajectory, the rollout therefore allows the policy to respond to object displacement, gripper mismatch, unexpected contact, and other state changes introduced by the recovery initialization.

Each scenario is evaluated under both open-gripper and closed-gripper initialization conditions. Unless otherwise specified, five trials are performed for each condition, resulting in ten recovery rollouts per scenario. The policy is allowed to interact with the environment for at most $H = 520$ control steps. A rollout is marked as successful when the environment emits the task-completion signal before the horizon is reached. If the task-completion condition is not reached within the available interaction budget, the rollout is marked as unsuccessful.

For the i-th rollout, we record the trajectory

$$\tau_i = \left(s_0^i, a_0^i, s_1^i, a_1^i, \ldots, s_{T_i}^i \right), \tag{17}$$

together with its task identifier, recovery level, gripper initialization, object perturbation, and binary outcome $y_i \in \{0, 1\}$. These episode-level records constitute the basic evidence used by the metric aggregation stage.

D.5 Metric Aggregation

The metric aggregation stage converts episode-level recovery outcomes into the benchmark statistics reported in the main paper. Each evaluated rollout is associated with a task instruction, a recovery scenario, a recovery difficulty level, a task-suite label, a gripper initialization condition, and a binary task outcome. The recorded trajectory additionally contains the observations, robot states, actions, and the temporal information required for phase-based analysis.

Episode-level outcome. For the i-th recovery rollout, we define a binary success indicator y_i according to the final task state:

$$y_i = \begin{cases} 1, & \text{if the original task goal is satisfied,} \\ 0, & \text{otherwise.} \end{cases} \tag{18}$$

A rollout is therefore considered successful only when the policy completes the original task after starting from the initialized recovery state. If the policy reaches the maximum interaction horizon without satisfying the task goal, the rollout is recorded as an unsuccessful recovery attempt.

Let $\mathcal{E}$ denote a set of evaluated recovery episodes under a particular evaluation condition. This set may correspond to all episodes of a model, a specific task suite, one recovery level, one gripper initialization condition, or any combination of these conditions. The Recovery Success Rate (RSR) is computed as

$$\text{RSR}(\mathcal{E}) = \frac{1}{|\mathcal{E}|} \sum_{i \in \mathcal{E}} y_i. \tag{19}$$

Thus, each rollout contributes one equally weighted success or failure sample. The overall RSR is computed over all valid recovery episodes, whereas the level-specific and suite-specific RSR values are obtained by restricting $\mathcal{E}$ to the corresponding subset.

For example, RSR_{L_3} is computed using only the recovery episodes assigned to object-state recovery. It measures how often the evaluated policy can complete the original task when the failure requires the restoration of a task-relevant object state. Likewise, the open-gripper and closed-gripper results are computed using the episodes associated with the corresponding initialization condition. This grouping allows the reported scores to distinguish overall recovery ability from performance under a particular failure difficulty or initialization condition.

Scenario- and task-level aggregation. A recovery scenario may be evaluated through multiple trials because the same initialized scene can be tested with different random object perturbations and gripper conditions. When the result is reported for one scenario, all valid trials associated with that scenario are included in the corresponding scenario-level recovery rate.

Task-level aggregation is performed over all recovery scenarios associated with the same original LIBERO task. Let $\mathcal{E}_k$ denote the set of all recovery episodes derived from task k. This set may contain multiple failure states, multiple perturbation instances, and repeated trials under different gripper initializations. The recovery rate of task k is therefore

$$R_k = \frac{1}{|\mathcal{E}_k|} \sum_{i \in \mathcal{E}_k} y_i. \tag{20}$$

In other words, R_k represents the recovery success rate over all failure scenarios related to task k, rather than the result of one representative failure scenario. When the two gripper conditions are analyzed separately, $\mathcal{E}_k$ is correspondingly restricted to the open-gripper or closed-gripper subset.

Recovery degradation. Recovery Degradation (RD) measures how much execution capability is lost after the execution enters a failure state. Each rollout is divided into three temporal phases: Pre-Failure, During-Failure, and Post-Failure. The Pre-Failure phase describes the execution before the first task-relevant deviation, whereas the Post-Failure phase describes the subsequent execution in which the policy must respond to the disrupted state and attempt to complete the original task.

Let m^i_{pre} and m^i_{post} denote the phase-specific execution performance of rollout i before and after the failure, respectively. These quantities are aggregated over the same evaluation subset:

$$M_{\text{pre}} = \frac{1}{|\mathcal{E}|} \sum_{i \in \mathcal{E}} m^i_{\text{pre}}, \qquad M_{\text{post}} = \frac{1}{|\mathcal{E}|} \sum_{i \in \mathcal{E}} m^i_{\text{post}}. \tag{21}$$

Here, M_{pre} is the average execution performance maintained before the failure occurs, and M_{post} is the average performance after the failure, when the policy must interpret the altered state, generate corrective actions, and resume the task. The two values are computed using matched task and scenario subsets so that their difference reflects the effect of failure rather than a change in task composition.

RD is then defined as

$$\text{RD} = 1 - \frac{M_{\text{post}} + \epsilon}{M_{\text{pre}} + \epsilon}, \tag{22}$$

where ϵ is a small constant for numerical stability. If the post-failure performance is close to the pre-failure performance, RD is close to zero, indicating that the policy retains most of its execution capability. If the post-failure performance approaches zero while the pre-failure performance remains positive, RD approaches one, indicating severe degradation. Therefore, RD measures relative capability retention rather than absolute recovery success. A policy may achieve a nonzero post-failure success rate while still obtaining a large RD if its pre-failure performance is substantially higher.

Recovery consistency. Recovery Consistency (RC) measures whether a policy exhibits similar recovery behavior across different tasks. For each task k, R_k is the recovery success rate computed over all failure scenarios and trials associated with that task. RC is defined as

$$\text{RC} = 1 - \text{Std}_k(R_k), \tag{23}$$

where Std_k denotes the standard deviation across task level recovery rates.

A high RC indicates that the policy's recovery performance varies little from one task to another. A low RC indicates that the policy performs well on some tasks but fails substantially on others. RC captures the stability of recovery behavior across tasks, rather than its absolute level. Consequently, RC is interpreted jointly with RSR. For example, a policy that fails on nearly all tasks may still obtain a relatively high RC if its failure rates are consistent, whereas a policy with both high RSR and high RC demonstrates stronger and more stable recovery capability.

Table 5: Inference configuration and LIBERO interface adaptation for the six evaluated policies.

Model	Policy type	Input interface	Precision	Control frequency	Chunk size	Training steps
OpenVLA-OFT	VLA	Instruction + image + proprioception	BF16	20 Hz	8/16/32	50K
GR00T-N1.5	VLA	Instruction + image + proprioception	BF16	20 Hz	8/16/32	30K
π_0	VLA	Instruction + image + proprioception	BF16	20 Hz	8/16/32	100K
π_0-Fast	VLA	Instruction + image + proprioception	BF16	20 Hz	8/16/32	30K
Wan2-Policy	WAM	Instruction + visual observation	BF16	20 Hz	8/16/32	60K
Cosmos-Predict2-Policy	WAM	Instruction + visual observation	BF16	20 Hz	8/16/32	50K

Stratified reporting. The final results are aggregated along several complementary dimensions. First, results are reported for each task suite to characterize the effect of task composition and scene diversity. Second, results are separated by the four recovery levels to show how performance changes as recovery requires increasingly deeper state reasoning. Third, open-gripper and closed-gripper trials are reported separately when analyzing the effect of the initial actuator condition. Finally, task-level recovery rates are used to compute RC, while phase-level performance statistics are used to compute RD.

All reported metrics are computed from the same episode-level records. The aggregation procedure does not depend on whether a policy is evaluated zero-shot, adapted with expert recovery trajectories, or provided with optional temporal context. These choices affect the observations, actions, and outcomes generated during each rollout, while the mapping from the resulting records to RSR, RD, and RC remains fixed.

E Experimental Implementation Details

E.1 Model Adaptation

We evaluate six representative manipulation policies on the LIBERO environment: OpenVLA-OFTKim et al. (2024), GR00T-N1.5Bjorck et al. (2025), π_0 Black et al. (2024), π_0-Fast Pertsch et al. (2025), Wan2-PolicyWan et al. (2025), and Cosmos-Predict2-PolicyAlhaija et al. (2025). All policies are deployed on the same Franka manipulation environment and receive the original LIBERO language instruction together with visual observations and robot state information. Since these policies use different input and action interfaces, we implement a lightweight environment adapter for each model.

The adapter converts LIBERO observations into the format expected by the corresponding policy, including image resizing, channel ordering, language-instruction formatting, proprioceptive-state normalization, and action normalization. The predicted actions are converted back to the LIBERO control space and executed by the simulator. The adapter does not modify the policy architecture or introduce additional task-specific information.

For OpenVLA-OFT, GR00T-N1.5, π_0, and π_0-Fast, the adapter provides the language instruction, the required camera observations, and the robot proprioceptive state to the corresponding VLA interface. The models output a chunk of low-level robot actions, which are executed sequentially before the next policy query.

Wan2-Policy and Cosmos-Predict2-Policy use a world-model-based action interface. For these models, the adapter organizes the current visual observation and language instruction according to the official model format and maps the predicted action representation to the seven-degree of freedom Franka control space and the gripper command used by LIBERO. All models are evaluated under the same simulator states, task instructions, recovery scenarios, timeout rules, and success criterion.

Table 5 summarizes the inference configuration used for each policy. Unless otherwise stated, inference is performed with bfloat16 precision on one NVIDIA H100 80GB GPU at a control frequency of 20 Hz.

All policies predict an 8-step action chunk per query. The predicted chunk is executed open-loop at 20 Hz, and the policy is re-queried after the chunk has been consumed, i.e., every eight control steps. In the chunk-size analysis of the main paper, only the number of executed actions per policy query is

Table 6: Backbones and action heads of the six evaluated policies. All policies share the same LIBERO training data, action space, and evaluation interface. Training steps are reported in Table 7.

Policy	VLM backbone	Auxiliary backbone	Action head
OpenVLA-OFT	Qwen3-VL-4B-Instruct	–	DiT-B
π_0	Qwen3-VL-4B-Instruct	DINOv2 ViT-S/14	DiT-B flow-matching
π_0-Fast	Qwen2.5-VL-3B-Instruct-Action	DINOv2 ViT-S/14	FAST tokenizer
GR00T-N1.5	Qwen2.5-VL-3B-Instruct	DINOv2 ViT-S/14	DiT-B flow-matching
Wan2-Policy	Qwen3-VL-4B-Instruct	Wan2.2-TI2V-5B	MLP
Cosmos-Predict2-Policy	Qwen3-VL-4B-Instruct	Cosmos-Predict2-2B	DiT-B flow-matching

varied from 4 to 32, while the observation interface, prediction horizon, and rollout protocol remain unchanged.

E.2 Unified Training of the Evaluated Policies

To ensure a controlled comparison, all six evaluated policies are trained within a single unified training framework Community (2026) and share the same data pipeline, action representation, and rollout interface. The four VLA policies (OpenVLA-OFT, π_0, π_0-Fast, and GR00T-N1.5) and the two world-model-based policies (Wan2-Policy and Cosmos-Predict2-Policy) differ only in their visual backbones and action heads, as summarized in Table 6.

All policies are trained on the initial LIBERO demonstrations, i.e., the union of the four training suites (`libero_spatial`, `libero_object`, `libero_goal`, and `libero_100`), comprising 6,193 episodes and 942,508 frames, where the LIBERO-100 suite jointly covers the `libero_10` (379 episodes) and `libero_90` (4,500 episodes) directories. Actions are 7-DoF delta end-effector poses with a gripper command, normalized with min–max statistics computed over the training set. Each policy receives synchronized third-person and wrist-camera images resized to 224×224 together with the robot proprioceptive state, and outputs an 8-step action chunk at the 20 Hz control frequency.

The flow-matching action heads used by π_0, GR00T-N1.5, and Cosmos-Predict2-Policy are 16-layer DiT-B transformers with a hidden size of 1024 and a cross-attention dimension of 2048, trained with 8 flow-matching steps and executed with 4 denoising steps at inference. π_0-Fast discretizes the action chunk with the FAST tokenizer and decodes action tokens autoregressively, and Wan2-Policy regresses the action chunk with an MLP head over world-model features.

Table 7 reports the per-policy optimization hyperparameters. All policies are optimized with AdamW (β_1=0.9, β_2=0.95, ϵ=1e−8, weight decay 1e−8). The learning rate follows a cosine schedule that decays to a minimum learning rate of 1e−6 after 5,000 linear warmup steps, and gradients are clipped at a global norm of 1.0. Learning rates are assigned per module group: 2.5e−5 for the base modules of OpenVLA-OFT, Wan2-Policy, and Cosmos-Predict2-Policy, and 3e−5 for those of π_0, π_0-Fast, and GR00T-N1.5; the VLM interface uses 1e−5 and the action head uses 1e−4 in all cases. Training uses a per-device VLA batch size of 16 for the VLA policies and 8 for the two WAM policies, with a single gradient-accumulation step, gradient checkpointing, and BF16 mixed precision, under a fixed random seed of 42. The VLA imitation loss has weight 1.0, and the VLM co-training loss on image–grounding corpora has weight 0.1 to preserve visual–language alignment during action training.

Following the official recipes of each policy family, all VLA is updated.

The training steps reported in Tables 5 and 7 denote the optimizer steps of the released checkpoint of each policy, selected by the standard LIBERO validation protocol of the corresponding recipe.

E.3 Recovery-Aware Fine-Tuning Configuration

The recovery-aware variants (*Combined*) in the main paper are obtained by jointly fine-tuning the policies on a mixture of the original LIBERO training data and the LIBERO-Recover expert recovery trajectories. Table 8 details the composition of the mixture, in which LIBERO-100 aggregates the

Table 7: Optimization hyperparameters of the six evaluated policies. All policies use AdamW, a cosine learning-rate schedule decaying to a minimum learning rate of $1e{-6}$ with 5,000 linear warmup steps, gradient clipping at 1.0, BF16 mixed precision, and seed 42.

Policy	Base LR	VLM LR	Action LR	Batch/GPU	Steps
OpenVLA-OFT	$2.5e{-5}$	$1e{-5}$	$1e{-4}$	16	50K
π_0	$3e{-5}$	$1e{-5}$	$1e{-4}$	16	100K
π_0-Fast	$3e{-5}$	$1e{-5}$	$1e{-4}$	16	30K
GR00T-N1.5	$3e{-5}$	$1e{-5}$	$1e{-4}$	16	30K
Wan2-Policy	$2.5e{-5}$	$1e{-5}$	$1e{-4}$	8	60K
Cosmos-Predict2-Policy	$2.5e{-5}$	$1e{-5}$	$1e{-4}$	8	50K

Table 8: Composition of the recovery-aware fine-tuning mixture. LIBERO-100 merges the `libero_10` and `libero_90` directories, and all components are sampled with equal weight.

Component	Episodes	Frames	Frame share
LIBERO-Spatial demonstrations	432	52,970	3.4%
LIBERO-Object demonstrations	454	66,984	4.3%
LIBERO-Goal demonstrations	428	52,042	3.3%
LIBERO-100 (`libero_10` + `libero_90`)	4,879	770,512	49.1%
Expert recovery trajectories	3,184	625,731	39.9%
Total	**9,377**	**1,568,239**	**100%**

`libero_10` and `libero_90` directories. All components are sampled with equal weights and no re-weighting, so the expert recovery trajectories account for 39.9% of the mixture frames and 34.0% of its episodes.

Table 9 summarizes the complete fine-tuning configuration. Fine-tuning is performed on 4 NVIDIA H20 (96 GB) GPUs with DeepSpeed ZeRO Stage 2 distributed training. The per-device batch size is 1 with 4 gradient-accumulation steps, yielding an effective batch size of 16, and training runs for 50,000 optimizer steps, corresponding to roughly half an epoch over the 1,568,239-frame mixture.

Unlike baseline training, where the full VLA is updated, the recovery-aware fine-tuning freezes only the vision encoder, and both the language model and the action head are updated, allowing the policy to adapt to the state distribution induced by failures. Checkpoints are saved every 10,000 steps and evaluated every 5,000 steps; the checkpoint at step 50,000 is used for all reported results.

The GR00T-N1.5 recovery-aware run adopts exactly the same protocol as the OpenVLA-OFT run, i.e., the identical data mixture, optimizer, schedule, number of steps, and freezing strategy, while its framework-specific settings are inherited from its initial-LIBERO recipe: the Qwen2.5-VL-3B backbone with the DINOv2 visual encoder, the DiT-B flow-matching action head, and a base learning rate of $3e{-5}$.

The *temporal* variants are trained with the same recipe, except that the pre-failure temporal history, i.e., the 8 observation frames preceding the current frame, is additionally provided to the policy as temporal context during both training and evaluation.

Table 9: Recovery-aware fine-tuning configuration shared by the *Combined* variants.

Item	**Setting**
Hardware	4× NVIDIA H20 (96 GB)
Distributed training	accelerate + DeepSpeed ZeRO Stage 2
Effective batch size	16 (4 GPUs × 1 per-device × 4 accum.)
Optimizer	AdamW (β_1=0.9, β_2=0.95, ϵ=1e−8, wd 1e−8)
Learning rate (base / VLM / action head)	2.5e−5 / 1e−5 / 1e−4
LR schedule	cosine decay to 1e−6, 3,000 warmup steps
Training steps	50,000 (≈0.5 epoch over the mixture)
Checkpointing	save 10,000 / eval 5,000 steps
Frozen modules	vision encoder only
Precision	BF16, gradient checkpointing, clip 1.0
Wall-clock time	≈5 h (≈1 h per 10,000 steps)
Random seed	42

F Failure Analysis

From the dataset construction process to the recovery experiments (Secs. 2.4 and 4), we observed several recurring patterns in the failures produced by the evaluated policies. These failures are not entirely random and can be roughly grouped into a small number of representative failure modes. They are related not only to imprecise low-level control, but also to the difficulty of updating the original action plan after the execution state has changed.

F.1 Failure to Verify Action Outcomes

The first issue is a mismatch between action outcomes and subsequent behavior: a policy may continue the original action sequence even when the preceding action has not achieved its intended result. Prior studies have identified related limitations in grounding action generation in visual observations. ReconVLA Song et al. (2026) reports that visual attention is insufficiently focused on manipulation targets, while Tsai et al. Tsai et al. (2026) show that directly providing target 3D locations to the action head improves spatial and task generalization. These findings suggest that policies may rely too heavily on familiar action patterns without sufficiently using current observations to adjust their actions. During failure-scenario collection, we observed two representative cases, shown in Figure 16.

In the first example, the instruction is `open_the_top_drawer_of_the_cabinet`, but the robot subsequently performs a placement motion near the plate. The video does not show an effective interaction with the drawer, and the later actions are not adjusted according to the state observed in the scene.

In the second example, the robot fails to grasp the black bowl but continues moving the empty gripper along the transport trajectory toward the plate. The problem is therefore not only the missed grasp. The later motion assumes that the bowl has already been acquired, although this prerequisite has not been satisfied.

These two cases show that the policy does not reliably check whether an action produced its intended result. When a grasp, placement, or contact fails, the policy may still follow the original plan, allowing the initial error to propagate through the remaining actions.

F.2 Insufficient Fine-Grained Control

The second type of failure occurs when the target object remains visible, but its position, orientation, or graspable region has changed. We show two L_2 failure cases produced by a model jointly trained on LIBERO and LIBERO-Recover, as reported in the main results. In the recovery scenarios, the

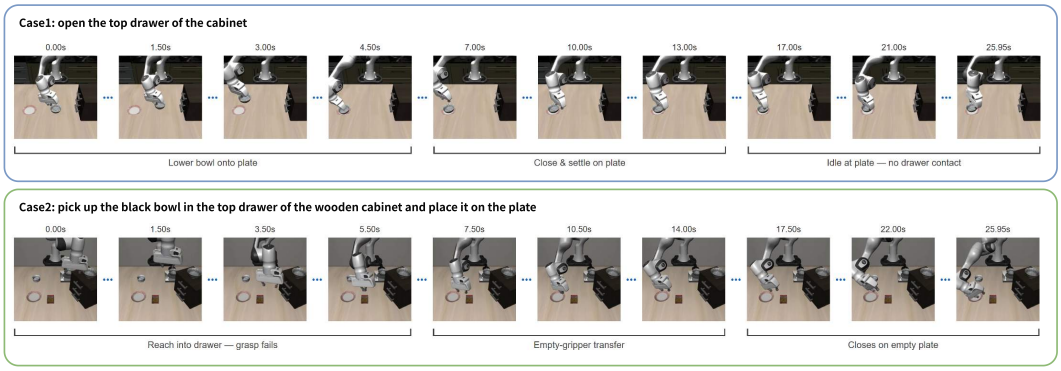


Figure 16: Representative failure cases in VLA rollouts.

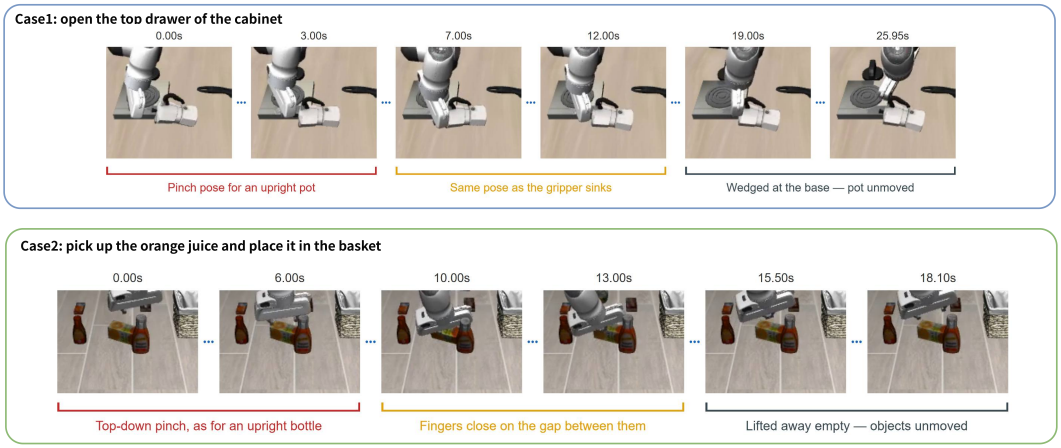


Figure 17: Two fine-grained grasping failures after object configuration changes.

model must select a specific grasping angle and approach direction according to the changed object configuration, which places greater demands on fine-grained control.

Figure 17 shows two representative examples. After the orange-juice container and the moka pot fall onto their sides, their height, orientation, and accessible grasp regions change. However, the policy continues to approach them with grasping motions that appear suitable for their original upright configurations. After several unsuccessful attempts, the gripper trajectory increasingly deviates from the target, but the policy does not identify a new grasp location or approach direction that could establish stable contact.

These failure cases show that detecting the target object is not sufficient for recovery. The policy must also determine which regions are graspable under the current configuration and from which direction the robot should approach. For an object that has been displaced or rotated, this decision often determines whether recovery can proceed.

F.3 Recovery Requires an Intermediate Action

Third, some failures cannot be resolved by directly repeating the nominal task action. The robot must first perform a prerequisite recovery action, such as repositioning an object, releasing the gripper, or removing an obstruction. We show two L_4 failure cases produced by a model jointly trained on LIBERO and LIBERO-Recover data. Although the model has been exposed to recovery trajectories, it still tends to execute the original task action directly in the recovery scenes instead of first restoring the object state required by the task.

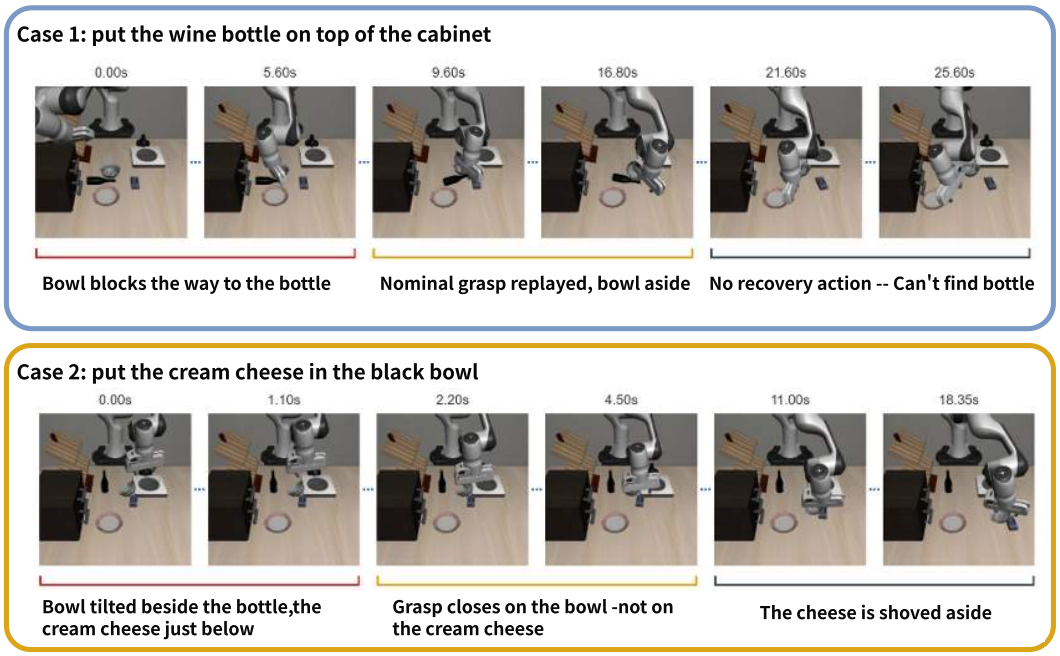


Figure 18: Examples of recovery failures that require prerequisite actions.

In the cream-cheese example in Figure 18(a), the initial failure leaves the robot holding the bowl instead of the cream cheese. At the beginning of the recovery rollout, the gripper is open, but the bowl remains close to the cheese. The model directly repeats the grasping motion toward the cheese and repeatedly contacts the bowl, without first moving the bowl aside or restoring a suitable spatial relation between the two objects.

In the wine-bottle example in Figure 18(b), a bowl blocks the robot's approach to the bottle. To complete the original task, the robot should first move the bowl aside and then replan the grasp for the bottle. Instead, the model directly executes the original bottle-grasping motion and makes additional contact with the bowl, causing the object configuration to move farther from a recoverable state.

The common problem in these two cases is not that the model fails to see the target object. Rather, it does not organize the actions in the order required for recovery. The model must first establish an intermediate state, for example by releasing an incorrectly held object, separating objects that interfere with each other, or clearing the approach path, before attempting the original task action again.

These cases show that L_3 recovery first requires the model to determine which states must be changed to restore the conditions for the original task. Without this prerequisite reasoning, directly repeating the original action may fail to complete the task and may introduce additional collisions or object displacement, making the scene increasingly different from familiar configurations and further complicating recovery. Robust behavior in such settings therefore requires the model to assess the current state and order its corrective actions before resuming the nominal task.

F.4 Recovery Degradation under Repeated Failures

To qualitatively illustrate the performance change measured by RD, we select a failed GR00T rollout on the task `stack_the_middle_black_bowl_on_the_back_black_bowl`. Along the original rollout, we save the simulator state every 20 control steps and rerun the same task from each recorded state. The arrows in the figure indicate the outcome of the rerun initialized from the corresponding state.

Figure 19 divides the rollout into three phases: Pre-failure, Failure, and Post-failure. The first two checkpoints are taken before the failure occurs, and the task can still be completed when rerun from

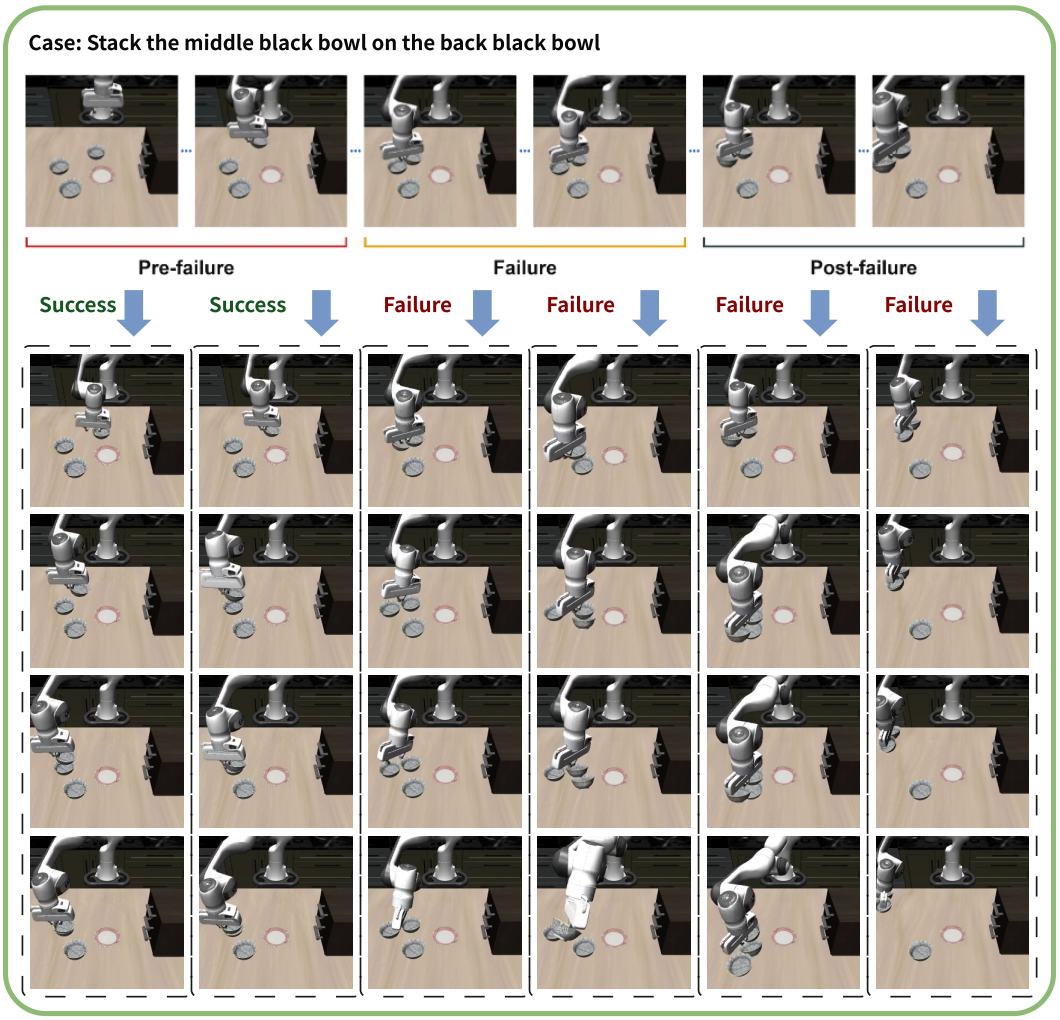


Figure 19: Qualitative illustration of recovery degradation after failure.

these states. After entering the Failure phase, the policy can no longer complete the original task from the resulting state, and the reruns remain unsuccessful in the Post-failure phase.

This case shows that failure changes more than the pose of an individual object. It can also alter the distribution of objects in the scene, their spatial relations, and the approach paths available to the robot. As execution continues, a scene that was initially solvable gradually becomes less familiar to the policy, which must reassess object relations and feasible ways of interacting with them.

This example provides a qualitative explanation for the further decrease in RD-related performance from During-Failure to Post-Failure reported in the main paper. The repeated interactions after failure do not return the scene to an easier configuration; instead, they introduce larger state shifts and stronger spatial reasoning demands for subsequent recovery. This case is intended to illustrate the behavior captured by RD rather than to estimate the metric independently.

G Challenges in Failure Recovery

The difficulty of recovery does not come only from changes in the scene. A failure may also change the gripper state, the end-effector pose, and the contact relationship between the robot and the objects.

G.1 Scene and Robot State Changes

Execution failures can change both the external scene and the robot's own interaction state. To illustrate these two types of changes, we select the collected failure cases.

Scene changes. Objects are displaced or rotated, which changes their support relations, contact relations, and approach paths. Although the task instruction and target object remain unchanged, the robot cannot directly reuse the manipulation strategy from before the failure. It must reassess the spatial relations between objects and identify a feasible approach region.

Robot state changes. The main change occurs in the robot itself. After failure, the end effector may remain in an awkward pose or approach the workspace boundary with a large orientation deviation. The objects may not have moved substantially, but the subsequent actions are no longer easy to execute smoothly, and the original action sequence may become ineffective.

This effect is also visible during teleoperation. In a small number of cases, the teleoperator does not immediately manipulate the target object, but first adjusts the end-effector position and orientation to bring the robot back to a more controllable configuration before continuing the recovery action. Recovery therefore sometimes requires restoring the robot's own operable state in addition to handling changes in the scene.

These cases show that post-failure changes include both visible changes in the scene and less obvious changes in the robot state. The former requires the policy to reinterpret the objects and their spatial relations, while the latter requires it to determine whether the current end-effector pose is still suitable for the next action. Ignoring either type of change can make subsequent actions difficult to execute.

G.2 Grasping under Different Object Configurations

The same task instruction may require different recovery actions under different post-failure states. To illustrate this point, we select the task "put the moka pot on the stove" and replay both an official LIBERO trajectory and our recorded recovery trajectories. Figure 21 shows the grasping segments from three replayed trajectories: Case 1 is taken from the official LIBERO Liu et al. (2023) data, whereas Cases 2 and 3 are from our teleoperated recovery data.

Although the final task goal is identical across the three trajectories, the position, orientation, and relative spatial relations of the moka pot during grasping are different. A grasp location or approach direction that is suitable for the standard initial configuration may therefore become ineffective under another post-failure configuration.

As shown in the figure, the robot must choose the end-effector approach according to the current pose of the moka pot rather than replaying the same grasping motion. Some configurations allow the gripper to approach from above, whereas others require a different wrist orientation or a lateral approach to establish stable contact.

This replay comparison shows that grasping in recovery cannot be determined solely by the task instruction and object category. The policy must also consider the current object configuration and the available space around it. Thus, even when the task goal remains unchanged, the effective recovery action may differ substantially from the action used in the original successful trajectory.

G.3 Implications for Recovery Performance

The cases above show that recovery differs from standard LIBERO evaluation in more than the initial object positions. The policy must determine which object relations remain usable after failure, whether the current end-effector pose is suitable for further interaction, and whether the original grasping action is still executable.

When an object is displaced or rotated, the policy must reconstruct its spatial understanding of the scene. When the robot is left in an unfavorable pose, it may also need to restore its own operable configuration first. These changes can occur together, so the policy cannot simply generate the

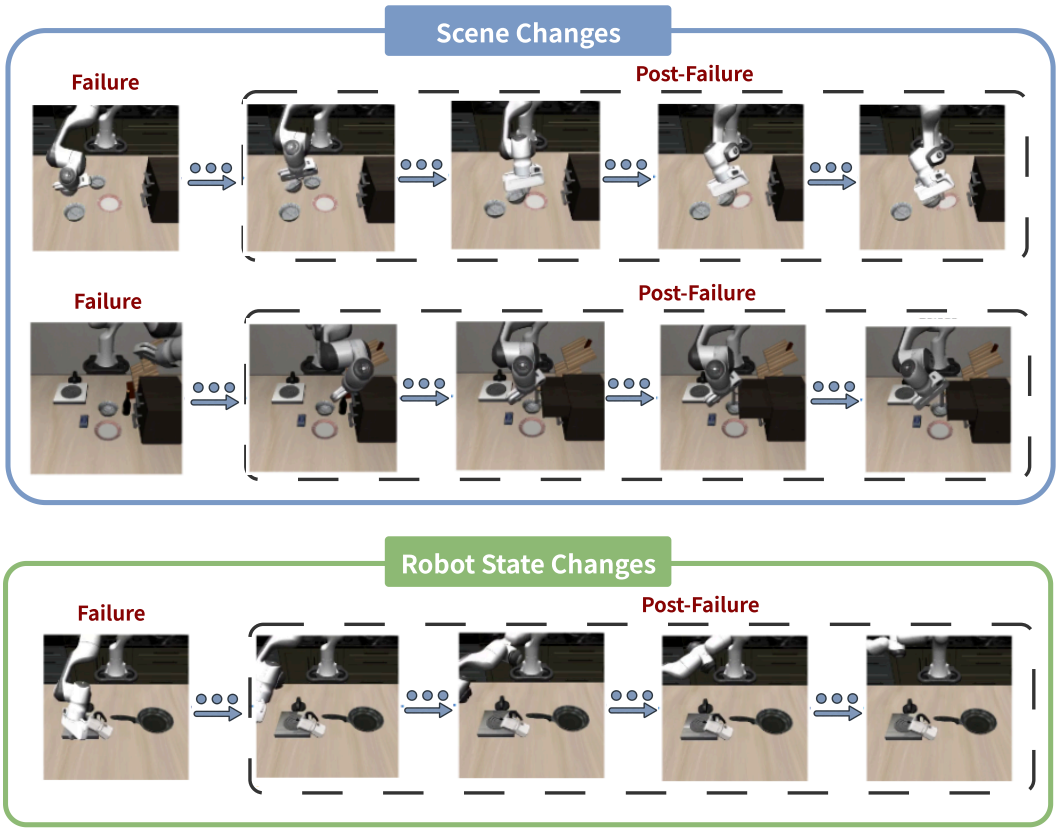


Figure 20: Scene and robot-state changes after failure. The first two cases show changes in object poses and relations between objects. The third shows an awkward robot configuration that hinders subsequent manipulation.

original action from the language instruction. It must reorder its actions according to the actual state.

This observation helps explain the performance difference between L_1/L_2 and L_3/L_4. Many L_1 failures and some L_2 failures can be handled by retrying or adjusting the next action, whereas L_3 and L_4 failures often require the robot to restore an object, robot, or environmental state before resuming the original task. These additional intermediate steps introduce more opportunities for failure during recovery.

The cases above do not separately quantify the contribution of scene changes, robot-state changes, and grasping precision to the final success rate. However, they are consistent with the lower RSR and higher RD observed in the main experiments. Recovery requires the policy to continuously check action outcomes and update its subsequent plan from the new state, rather than replaying the action sequence from a successful trajectory.

H Limitations and Discussion

LIBERO-Recover is currently constructed in simulation, with failure localization and categorization assisted by a vision-language model and verified by human reviewers. The results may therefore not fully reflect sensor noise, control errors, or hardware limitations in physical robots. In addition, the naturally imbalanced failure distribution leads to fewer examples at the higher recovery levels.

Nevertheless, the quantitative results and qualitative cases consistently show that recovery requires the policy to reinterpret object states, spatial relations, and robot pose after failure rather than simply

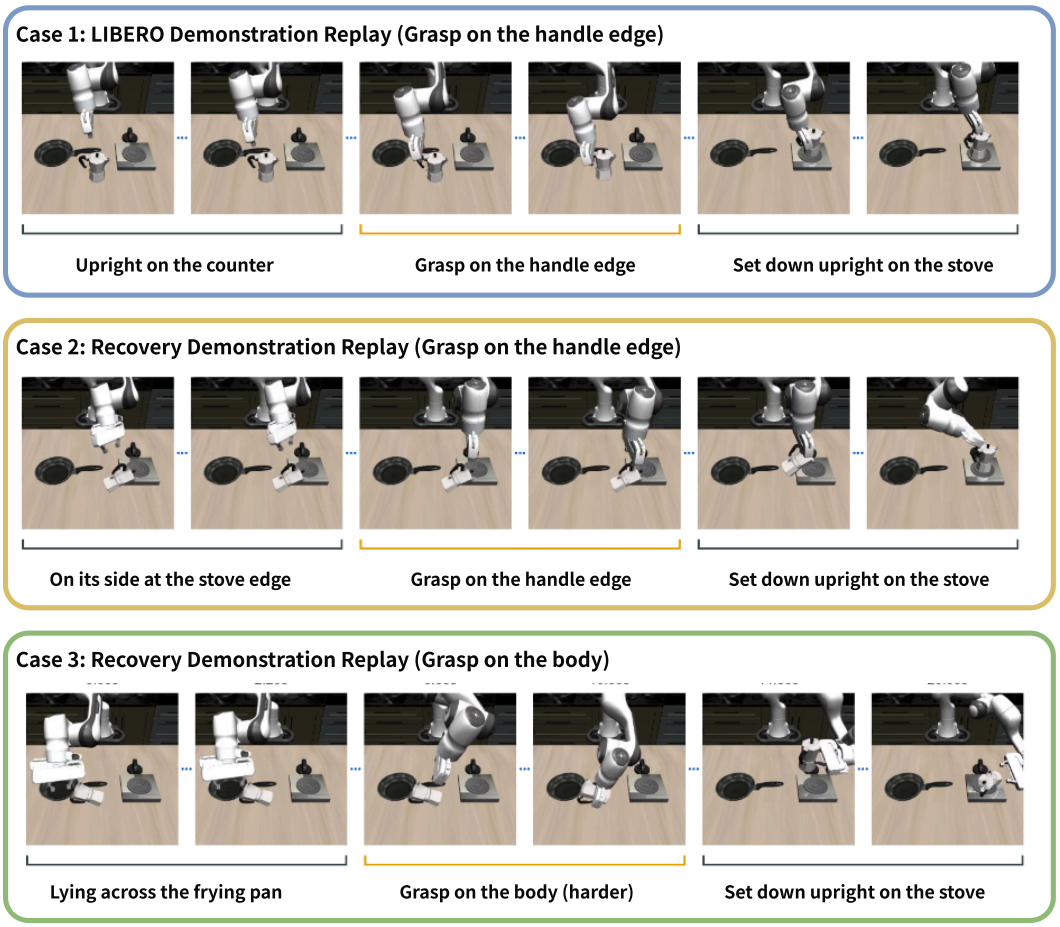


Figure 21: Replay of recorded demonstrations: Case 1 uses an official LIBERO demonstration, while Cases 2 and 3 use our teleoperated recovery demonstrations.

repeat the original action. Future work will validate these observations on physical robots and extend the evaluation to different robot arms and multi-arm cooperative manipulation.