RepairVLA: Enhancing Vision-Language-Action Models with Failure Repair Mechanisms

Anonymous ACL submission

Abstract

Vision-Language-Action(VLA) models have emerged as a promising foundation for general-purpose robotic manipulation. However, they demonstrate limited robustness in long-horizon, complex missions. Due to rigid sequential dependencies, a single execution error disrupts the task chain by invalidating future preconditions, leading to irreversible drift into out-of-distribution regions. Existing methods attempt to enhance VLA reasoning but predominantly focus on initial task decomposition and planning, leaving the agent unable to perceive or rectify intermediate execution failures. To address this, we introduce **RepairVLA**, a novel architecture that establishes an "Execute-While-Repairing" paradigm. We propose a "**Decompose-Execute-Check-Repair**" Chain-of-Thought(CoT) to endow the agent with the capability to detect runtime failure and dynamically initiate corrective maneuvers. Crucially, to enable the learning of such robust behaviors despite the scarcity of negative samples, we design an **automated data generation pipeline** that leverages Vision-Language Models(VLMs) to analyze error causes and synthesize repair trajectories in simulation. Experiments on long-horizon benchmarks show that RepairVLA significantly improves success rates(+9.9%) through autonomous error recovery. Furthermore, our iterative training results demonstrate that the model can evolve by learning from its own execution errors, leading to a robust, self-correcting policy for multi-stage manipulation tasks.

1 Introduction

Vision-Language-Action(VLA) models, such as RT-2(Brohan et al., 2023a), OpenVLA(Kim et al., 2024), π_0(Black et al., 2026), and the hierarchical $\pi_{0.5}$(Intelligence et al., 2025), have established a new standard for general-purpose robotic manipulation. By combining the semantic reasoning of Large Language Models(LLMs) with visual encoders and continuous action heads(Driess et al., 2023), these models demonstrate strong capabilities in instruction following and zero-shot generalization. Trained on massive datasets like Open X-Embodiment(Collaboration et al., 2025), VLAs map high-level instructions directly to low-level controls, shifting the focus of Embodied AI from specialized policies to foundation models.

However, deploying these models in long-horizon, multi-stage missions remains challenging(Ahn et al., 2022). Benchmarks like LIBERO(Liu et al., 2023a) and RLBench(James et al., 2019) show that performance drops sharply as task length increases. The core challenge is the strict sequential dependency of these missions: a failure at any intermediate stage disrupts the preconditions for subsequent steps. Consequently, a single error propagates through the execution chain, driving the agent's state away from its training distribution and leading to inevitable failure(Huang et al., 2022).

We therefore propose a "Decompose-Execute-Check-Repair" closed-loop paradigm(Figure 1): the agent verifies state consistency in real-time and switches to a repair policy upon detecting deviations, actively preventing error propagation. However, realizing this paradigm is hindered by the lack of paired failure-recovery samples in standard expert-dominated benchmarks(Ross et al., 2011; Laskey et al., 2017). To overcome this, we introduce **RepairVLA**, which combines a runtime inference architecture with an automated failure-repair data pipeline.

First, we propose a **Dual-Stream Inference** architecture that unifies two distinct pathways: a **Base Stream** for normal task progression and a **Repair Stream** for failure recovery. Following a **Decompose-Execute-Check-Repair** paradigm, the system employs a *Visual Monitor* to verify subtask completion in real-time, activating

the Repair Stream only when deviations are detected. **Second**, to enable the learning of these recovery behaviors, we develop an **Automated Failure-Recovery Data Pipeline**. We leverage a VLM to generate executable code that invokes motion primitives, efficiently synthesizing "failure-recovery" training pairs without manual intervention. Empowered by automated repair data generation, this closed-loop architecture, RepairVLA achieves a continuous loop of executing, checking, and repairing, enabling autonomous error recovery.

Our contributions are summarized as follows:

1. **Closed-Loop Architecture**: RepairVLA embeds a Decompose-Execute-Check-Repair loop that transitions from open-loop prediction to feedback-aware control.

2. **Automated Data Pipeline**: A fully automated pipeline generates failure-repair trajectories without manual labeling.

3. **Strong Results**: RepairVLA surpasses π_0 and other baselines on LIBERO and RLBench, and further improves through iterative self-training.

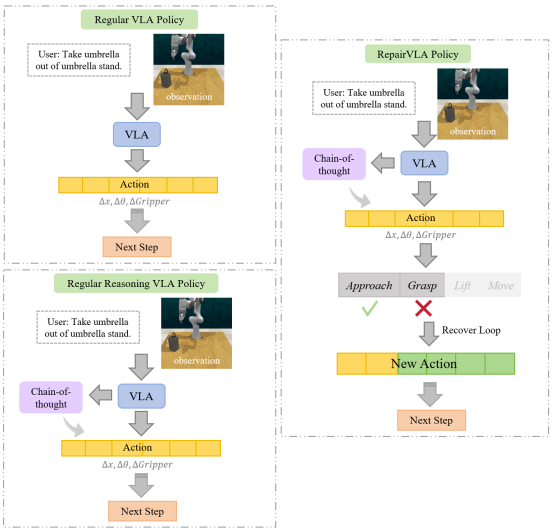


Figure 1: **Evolution from Open-Loop Prediction to Closed-Loop Recovery.** (**left**)Standard VLAs directly map instructions to actions in a black-box manner. (**middle**)Reasoning VLAs improve upon this by leveraging Chain-of-Thought(CoT); however, they remain open-loop. Once a plan is generated, the agent blindly executes it without feedback, causing errors to propagate. (**right**) RepairVLA(Ours) introduces a closed-loop "**Decompose-Execute-Check-Repair**" paradigm. By integrating a real-time monitor and a Recover Loop, it not only plans but also actively detects failures and synthesizes corrective "New Actions" to dynamically repair the mistake.

2 Related Work

VLAs have recently demonstrated remarkable potential in bridging multi-modal perception with robotic control. However, the lack of intrinsic error-recovery capabilities remains a critical barrier to autonomous correction. To contextualize our work, We categorize VLA literature into Non-Reasoning and Reasoning architectures(Ma et al., 2025), and identify the data bottleneck that limits model robustness.

2.1 Non-Reasoning VLAs

Early VLA primarily adopt end-to-end imitation learning(Brohan et al., 2023b; Reed et al., 2022). Subsequent works like A2C2(Sendai et al., 2025) and RT-Trajectory(Gu et al., 2023) improve performance by adding auxiliary heads for trajectory smoothing. While effective against execution jitter, these reactive policies lack the semantic introspection required to diagnose or rectify logical failures.

Consequently, these purely reactive models lack the semantic introspection required to diagnose errors, rendering them incapable of recovery.

2.2 Reasoning VLAs

Recent research integrates CoT or high-level planning to improve long-horizon consistency(Liang et al., 2023; Belkhale et al., 2024).

2.2.1 Open-Loop Reasoning VLAs

These methods leverage LLMs to generate high-level plans before execution. Frameworks like Embodied-CoT(Zawalski et al., 2025), CoT-VLA(Zhao et al., 2025), and ManualVLA(Gu et al., 2025) utilize reasoning steps or manuals to guide policies. Notably, ThinkAct(Huang et al., 2025) compresses reflection into latent variables to modulate low-level policies, while RAD(Clark et al., 2025) extracts reasoning traces from human videos to guide policy learning.

Despite better planning, these methods treat execution as a black box. The reasoning acts as a feed-forward signal without immediate feedback, preventing the system from rectifying physical errors during execution.

2.2.2 Reflective VLAs

To address execution failures, recent works attempt to introduce recovery mechanisms.

LITEN(Shah et al., 2025) and RECAP(Physical Intelligence et al., 2025) attempt post-hoc analysis or human-teleoperated corrections, while ViFailback(Zeng et al., 2025)utilizes visual symbols to diagnose failures.

However, these methods rely on slow replanning, expensive human intervention, or non-executable diagnostic symbols. They fail to treat recovery as an intrinsic, real-time capability, the realization of self-correcting systems.

2.3 Datasets and Benchmarks for Robustness

The scarcity of failure-repair data remains a fundamental constraint(Wang et al., 2021).

Large-scale datasets like Open-X Embodiment(Collaboration et al., 2025; Khazatsky et al., 2025) consist exclusively of successful demonstrations. Similarly, synthetic frameworks ANYTASK(Gong et al., 2025) focus on generating optimal trajectories, providing no signal on how to recover from errors. Despite RoboFail(Liu et al., 2023b) and RoboFAC(Lu et al., 2025) provide datasets containing failure instances and detailed taxonomies, they lack the corresponding corrective trajectories.

Consequently, the field lacks a unified resource that pairs failure states with valid repair actions, creating a data gap that prevents the large-scale training of autonomous self-correcting agents.

3 Method

To achieve robust long-horizon manipulation, we must shift from the traditional open-loop execution paradigm to a closed-loop framework that dynamically detects and repairs errors during deployment. To realize this "**Decompose-Execute-Check-Repair**" cycle, we require a system capable of autonomously synthesizing corrective actions to restore task progress upon failure. Specifically, we propose **RepairVLA**, a **Dual-Stream Inference** architecture trained via an automated data synthesis pipeline. The Dual-Stream unifies a **Base Stream** for normal execution and a **Repair Stream** for error correction. This approach enables the agent to autonomously detect failures and learn active recovery strategies without relying on dense human intervention. As illustrated in Figure 2, our framework consists of three key components: a Dual-Stream Inference loop for real-time monitoring and repair plan generation, an automated pipeline for failure-recovery data synthesis, and an *Iterative Training* strategy for self-improvement.

Algorithm 1 Task execution with failure recovery

Require: Global instruction I, initial observation O_0
T_i: i-th subtask instruction; V_i: verification condition for T_i
// Decompose: Split global task into subtasks with verification conditions
$(T_1, \ldots, T_n), (V_1, \ldots, V_n) \leftarrow$ VLM.Decompose(I)
$O \leftarrow O_0, i \leftarrow 1$
while $i \leq n$ **do**
  // Execute: Run current subtask
  $A \leftarrow$ VLA.Forward$(O, T_i), O' \leftarrow$ Env.Step(A)
  // Check: Verify subtask completion
  if VLM.VQA(O', V_i) **then**
    $O \leftarrow O', i \leftarrow i + 1$
  else
    // Repair: Start repair at failure node and execute the new plan
    $(T'_1, \ldots, T'_n), (V'_1, \ldots, V'_n) \leftarrow$ VLM.Repair(O', I)
    $i \leftarrow 1, O \leftarrow O', n \leftarrow n'$
  end if
end while
return success

3.1 Core Architecture: Dual-Stream Inference

To realize the "Decompose-Execute-Check-Repair" paradigm, we implement a *Reasoning-Acting Loop* that couples a VLM for high-level planning and execution monitoring with a VLA model for low-level physical execution (Black et al., 2026; Chen et al., 2025). Specifically, the VLM serves as the high-level controller: it decomposes the instruction into subtasks to guide the VLA, and continuously monitors the execution state via feedback. Based on this feedback, the VLM assesses subtask completion and dynamically decides whether to proceed or to generate a repair plan upon failure. Meanwhile, the VLA functions as the executor: conditioned on real-time observations and the subtask instruction issued by the VLM, it produces actions and returns execution feedback to close the loop. The complete inference procedure is formalized in Algorithm 1, which specifies the Decompose-Execute-Check-Repair closed-loop execution with failure recovery.

Overall, the VLA is reliable for in-distribution skill execution, and the VLM serves as a task-level controller that decomposes and reshapes instructions into executable subtasks within the VLAs capability set, while using feedback to decide when to proceed or invoke repair.

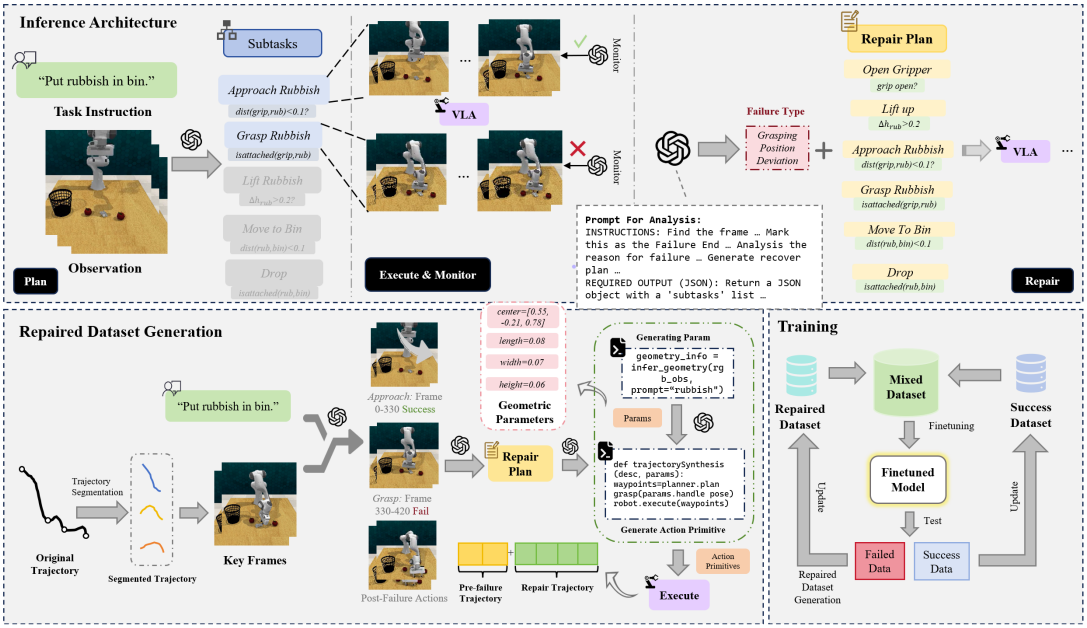


Figure 2: **The Overview of RepairVLA Framework.** The system consists of three key components: **(Top) Dual-Stream Inference Architecture:** A VLM serves as planner and real-time *Monitor*, decomposing tasks and evaluating execution. On failure, it diagnoses and generates a *Repair Plan* to guide the VLA. **(Bottom-Left) Automated Data Generation Pipeline:** To mitigate data scarcity, we segment failure trajectories, extract key frames, and use a VLM to infer parameters and generate code that invokes *Action Primitives* to synthesize repair trajectories. **(Bottom-Right) Iterative Training Strategy:** A self-improvement loop fine-tunes the VLA on mixed success and repair data. The updated model is redeployed to collect new failures, which are repaired and reused for training, enabling continuous evolution of robustness.

3.1.1 Subtask Decomposition

Given a natural language instruction $\mathcal{L}$, the VLM functions as a high-level planner. It decomposes the instruction into a sequence of subtasks $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$. For each subtask, the VLM generates a semantic description T_i alongside a set of verification conditions V_i, which is state-based criteria defining success, such as "gripper is closed" or "object is inside the bin". These descriptions and conditions are passed directly to the *Visual Monitor*, serving as the standards for real-time success detection.

3.1.2 Execution & Real-time Monitoring

During the execution phase, the VLA acts as the low-level controller, generating a sequence of actions based on current visual observations O_t to fulfill the subtask T_i. In parallel, the VLM serves as a *Visual Monitor*, utilizing the previously defined verification conditions V_i to evaluate the environment via a Visual Question Answering(VQA) interface(Agrawal et al., 2016) upon the completion of each generated action sequence. This step compares the current state against V_i to determine whether T_i has been successfully accomplished or if a deviation has occurred(Yu et al., 2023).

3.1.3 Failure Diagnosis & Repair

When the monitor detects that the current subtask is not achieved(or that an execution deviation occurs), the system switches from the base stream to the *Diagnosis & Repair* stream. The VLM analyzes the current observation O_t to infer the failure type, and synthesizes a repair plan as a short sequence of corrective natural-language sub-instructions. These repair instructions are then fed sequentially to the VLA, which generates corresponding actions to correct the error and restore the system to a valid state for continued execution. Our framework targets general cross-task failures(e.g., slippage, grasp deviation, misidentification) rather than task-specific errors, enabling broad applicability across manipulation scenarios.

3.2 Automated Failure-Recovery Data Generation Pipeline

The RepairVLA framework requires a policy robust to both nominal execution and recovery scenarios. However, standard VLAs generally lack this repair capability. This limitation stems from the nature of existing datasets, which are

Algorithm 2 Automated Repair Data Generation

Require: Failure trajectory set D_{fail}, task instruction set $\mathcal{I}$
$D_{\text{repair}} \leftarrow \emptyset$
for each failure trajectory $T_{\text{fail}} \in D_{\text{fail}}$ **do**
  $I \leftarrow$ corresponding task instruction of T_{fail}
  // Segment: Split trajectory into subtasks and locate the failed subtask k
  $[T_1, \ldots, T_k, \ldots, T_n]$, key_frames $\leftarrow$ TrajectorySeg(T_{fail}, I)
  $T_{\text{pre}} \leftarrow [T_1, \ldots, T_{k-1}]$
  // Plan: Generate repair plan
  repair_plan $\leftarrow$ VLM.Plan(key_frames, I)
  // Execute: VLM generates executable programs with parameters for the robot to output repair trajectory
  $T_{\text{repair}} \leftarrow$ Execution(repair_plan)
  // Assemble: Concatenate the pre-failure and repair trajectory
  $T_{\text{full}} \leftarrow$ Concatenate$(T_{\text{pre}}, T_{\text{repair}})$
  $D_{\text{repair}} \leftarrow D_{\text{repair}} \cup \{T_{\text{full}}\}$
end for
return D_{repair}

dominated by successful trajectories, and failure datasets typically provide only high-level labels without the corrective action trajectories needed for training(Liu et al., 2023b; Lu et al., 2025). To resolve this scarcity, we introduce a scalable **Automated Failure-Recovery Generation Pipeline**. This pipeline, visualized in the bottom-left of Figure 2, leverages a VLM to identify failures in natural rollouts and synthesize corresponding repair trajectories via a programmatic approach(Kim et al., 2024). By automating both annotation and synthesis, we efficiently generate paired failure-repair data without manual intervention. The complete pipeline is formalized in 2.

3.2.1 Automated Annotation

To ensure failure data diversity and validity, we first deploy standard VLAs on common benchmarks (RLBench (James et al., 2019), LIBERO (Liu et al., 2023a)) to capture a broad range of natural failure modes. We split the failure annotation into two stages: candidate subtask segmentation and failure identification.

We formalize the segmentation as follows. Let a raw execution trajectory be $\tau = \{(\mathbf{x}_t, g_t)\}_{t=0}^{T}$, where $\mathbf{x}_t = (\mathbf{p}_t, \mathbf{R}_t)$ is the end-effector pose (3D position/orientation) and $g_t \in \{0, 1\}$ is the gripper state. To measure movement intensity, we compute the joint motion magnitude $u_t = \sqrt{\|\mathbf{v}_t\|_2^2 + \lambda\omega_t^2}$, with $\mathbf{v}_t$ (translational velocity), ω_t (angular velocity), and λ (weighting factor). Candidate boundaries B are identified via three kinematic triggers: Dwell Detection (motion pauses with $u_t < \varepsilon_v$ for H duration), Jerk Peak Detection (local jerk maxima exceeding ε_j), and Gripper Events (logical transitions where $g_t \neq g_{t-1}$). Boundaries are pruned by merging points within temporal window δ and enforcing minimum segment length $L_{\min}$, yielding subtask segments $\mathcal{S} = \{s_1, s_2, \ldots, s_K\}$; boundary frames serve as key frames for critical state transitions.

Next, the VLM performs two consistency checks on these segments: kinematic logic validation (ensuring physically coherent action sequences, e.g., no teleportation, gripper commands preceding object interaction) and visual semantic verification (using VLM-VQA on key frames to confirm execution aligns with semantic intent). Only segments passing both checks are retained. Guided by the predefined error taxonomy 1, the VLM maps each failed instance to its category. We use general cross-task failure types (not task-specific) that cover most recoverable failures in daily robotic manipulation.es rather than task-specific errors, covering most recoverable failures in daily robotic manipulation.

3.2.2 Repair Data Synthesis

Textual repair plans must be converted into physical trajectories for training. To achieve this, we implement a Code-based Repair Synthesis module, which operates through two distinct code generation phases: parameter extraction and action primitive generation. Upon confirming a failure, the VLM generates executable Python code to invoke visual tools to extract a set of predefined geometric parameters(Lin et al., 2025). These parameters explicitly quantify the scene geometry and kinematics, including object pose and dimensions(e.g., center, bounding box, radius) as well as articulation properties(e.g., hinge axis, sliding direction). Then, utilizing these parameters, the VLM generates a subsequent script that calls robot motion primitives(e.g., `move_to()`, `grasp()`) based on these parameters. These primitives are executed by a Python interpreter to drive a motion planner, ultimately synthesizing the repair trajectories.

3.3 Model Training

We incorporate failure-repair capabilities into the VLA policy through fine-tuning on a mixed dataset that integrates synthesized repair trajectories with standard expert demonstrations.

3.3.1 Dataset Construction

We organize the collected data into a unified format consisting of triples (S_t, L_t, A_t). Here, S_t represents the sequence of visual observations and robot states, L_t denotes the language instruction(which switches to the repair instruction dur-

Table 1: **Taxonomy of Failure Modes.** A summary of failure types autonomously categorized by the VLM from natural rollouts. Errors are divided into recoverable and irrecoverable categories based on the feasibility of a physical repair action.

Category	Specific Failure Modes
Recoverable Errors	Target Misidentification Grasping Direction Deviation Gripper Orientation Error Object Slippage / Drop Motion Planning Failure
Irrecoverable Errors	Target Out Of Workspace Object Physical Damage

ing correction phases), and A_t is the target action. This uniform representation allows us to seamlessly mix standard demonstration data with our generated failure-recovery data. In failure states S_{fail}, corrective actions A_{repair} require both the robot state S_{fail} and the repair instruction L'_t, as validated by our ablation studies showing that training with full task trajectory context outperforms isolated repair segments.

3.3.2 Iterative Training

We fine-tune the backbone model on this unified dataset using Conditional Flow Matching, with the training objective of aligning the model-predicted conditional vector field $\mathbf{v}_t$ with the target optimal transport vector field $\mathbf{u}_t$ to generate smooth repair trajectories from failure states. The training loss is formulated as $\mathcal{L}_{\text{CFM}} = \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_1,\mathbf{u}_t}\|\mathbf{v}_t(\mathbf{x}_t, t, \mathbf{c}) - \mathbf{u}_t\|_2^2$, where $\mathbf{c}$ denotes task condition and failure observation. This explicitly teaches the model that in a failure state S_{fail}, the optimal policy is the corrective action A_{repair}. To further enhance robustness, we implement an Iterative Training loop: the trained model is re-deployed to collect new failure modes, which are processed by our pipeline to generate fresh repair data and added to the training set. This cycle allows continuous improvement of robustness in long-horizon tasks. We adopt GPT-5 for high-level VLM tasks.

4 Experiment

In this section, we empirically evaluate RepairVLA to assess its capability in performing autonomous error recovery during long-horizon tasks. Our evaluation is centered around the following three research questions(RQs): **RQ1(Main Results):** Does RepairVLA outperform state-of-the-art VLA baselines on long-horizon manipulation tasks? **RQ2(Ablation Study):** How does the utilization strategy of synthesized repair data affect the robustness of the learned policy? **RQ3(Evolution):** Can the model achieve continuous self-improvement through an iterative “execution-repair-retraining" loop?

4.1 Experimental Setup

To assess the robustness and repair capability of RepairVLA, we conduct experiments on two mainstream robot learning simulation platforms: **RLBench** and **LIBERO**.

Benchmarks and Tasks

- **RLBench**(CoppeliaSim/PyRep): six diverse tasks (*Put Rubbish In Bin*, *Take Umbrella*, *Close Laptop*, *Stack Wine*, *Take Frame Off Hanger*, *Phone On Base*; abbreviated in tables).
- **LIBERO**: (MuJoCo): 10 tasks (T0-T9) evaluating long-horizon skill diversity.

Robot Embodiment All experiments are conducted on a 7-DoF Franka Emika Panda manipulator equipped with a parallel-jaw gripper.

Observation and Action Space We employ a multi-view camera setup to provide a comprehensive observation of the workspace. This configuration consists of a static third-person view and a dynamic wrist-mounted view, both of which are processed as RGB images.

The proprioceptive state is represented as a unified 8-dimensional end-effector vector $s_t \in \mathbb{R}^8$, comprising the 3D world coordinates, 3D orientation (axis-angle representation), and 2D gripper status.

We utilize a 7-DoF End-Effector Control Scheme. The action vector a_t consists of end position displacement $(\Delta x, \Delta y, \Delta z)$, rotational changes $\Delta\theta$(represented as quaternions), and the discrete gripper state.

Baselines We compare RepairVLA against the π_0-base and OpenVLA-OFT model. To ensure a fair comparison, the baseline employs the identical model architecture but is fine-tuned on successful expert demonstrations from RLBench and LIBERO, without the inclusion of synthesized repair data.

Training details The model is fine-tuned on a cluster of 8× NVIDIA A800(80GB) GPUs using a per-device batch size of 16. We use the AdamW optimizer with a base learning rate of 1×10^{-4}, $\beta_2 = 0.95$, and a weight decay of 1×10^{-10}. To ensure stability during the iterative update phase, a linear warmup is applied for the first 1000 steps.

4.2 Experimental Analysis

4.2.1 RQ1: Comparison with State-of-the-Art Baselines

To answer RQ1, we evaluate the performance of RepairVLA against π_0-base across the LIBERO-Goal and RLBench benchmarks. For the LIBERO-Goal suite, we specifically compare two data sampling strategies to assess the impact of repair data distribution: **Uniform Sampling**, and **Active Sampling**. As summarized in Table 2, RepairVLA consistently outperforms the baseline, demonstrating the effectiveness of incorporating repair data into policy learning.

Repair data improves robustness. RepairVLA-π_0 (Active) raises LIBERO-Goal average success from 86.1% to 90.9%, with gains on hard tasks like T3/T6 (from 66.7% to over 80%). The same trend holds for OpenVLA-OFT and on RLBench (+9.9%), confirming that the model learns recovery instead of terminating on failure.

Active vs. uniform mixing. Active sampling reweights underperforming tasks but yields marginal overall gain (0.909 vs. 0.906) at the cost of non-monotonic changes and budget competition. We therefore use uniform mixing as the default for simplicity and reproducibility, consistent with the phenomenon that cumulative expansion mitigates budget-induced forgetting at the cost of a growing dataset.

4.2.2 RQ2: Impact of Utilization Strategy of Repair Data Mixture

To address RQ2, we evaluate the influence of repair data through two dimensions: **mixture ratio** and **temporal context**. We fine-tune the backbone model with varying proportions of repair data and, specifically on LIBERO, compare *Standalone Repair Segments* against *History-Conditioned Trajectories* to assess the necessity of failure history. Results indicate that a moderate injection of repair data, combined with failure history, maximizes policy robustness.

Table 2: **Comparison with baselines.** LIBERO-Goal (top), sampling ablation (middle), and RLBench (bottom).

(a) LIBERO-Goal (Success Rate)

Method	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	**Avg**
π_0	0.860	0.980	0.860	0.667	0.973	0.853	0.667	**1.000**	0.927	0.827	0.861
RepairVLA-π_0	0.920	**1.000**	0.873	0.833	0.940	0.907	0.807	0.993	0.947	0.867	0.909
OpenVLA-OFT	**0.960**	**1.000**	0.920	0.890	**0.990**	**1.000**	**0.940**	**1.000**	**1.000**	0.670	0.937
RepairVLA-OFT	0.930	**1.000**	**0.950**	**0.920**	0.940	0.980	**0.940**	**1.000**	0.980	**0.880**	**0.952**

(b) Uniform vs. Active sampling (RepairVLA-π_0, 2% repair data)

Strategy	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	**Avg**
Uniform	**0.933**	0.953	0.853	0.813	0.933	**0.947**	**0.840**	0.980	0.960	0.847	0.906
Active	0.920	**1.000**	**0.873**	**0.833**	**0.940**	0.907	0.807	**0.993**	0.947	**0.867**	**0.909**

(c) RLBench (Success Rate)

Method	Rubbish	Umbrella	Laptop	Wine	Frame	Phone	**Avg**
π_0	0.800	0.460	0.180	0.060	0.400	0.100	0.333
RepairVLA	**0.920**	**0.640**	**0.320**	**0.170**	**0.420**	**0.120**	**0.432**

Mixture ratio ablation. A small repair fraction (2% on LIBERO, 5% on RLBench) yields the best robustness, suggesting sparse corrective trajectories suffice to regularize the policy. Higher ratios degrade performance, especially on precision tasks like *Stack Wine*, due to distribution shift causing hallucinated or unnecessary recovery.

Ablation of Temporal Context of Repair Data: Temporal context is essential for effective failure recovery. Table 4 highlights the critical role of temporal context. The history-conditioned models consistently outperform the standalone segments approach. Notably, using isolated repair segments fall slightly below the baseline despite gains in specific subtasks. This indicates that the policy relies on the failure context to correctly identify the need for repair, rather than simply memorizing repair motions.

4.2.3 RQ3: Evolution via Iterative Training

For RQ3, we examine the system's ability to self-improve on LIBERO-Goal. We adopt an iterative framework where the initial model is deployed to collect failure trajectories, which are then converted into repair data via our automated pipeline for subsequent fine-tuning. As summarized in Figure 3, the results confirm that this iterative self-correction mechanism significantly enhances policy robustness compared to the static baseline.

Efficacy of Iterative Training. The experimental results validate that our iterative training strategy effectively drives policy evolution. As shown in 3,the average success rate increases steadily

Table 3: **Ablation Study on Repair Data Mixture Ratios.** We train the policy with varying percentages of synthesized repair data mixed into the standard expert demonstrations(Baseline 0%). The results on LIBERO-Goal(top) and RLBench(bottom) indicate that a small fraction of repair data yields the most significant robustness gains, while excessive mixing degrades performance.

(a) **LIBERO-Goal Results**(Success Rate)

Mix Ratio	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	Avg
0%	0.860	0.980	0.860	0.667	0.973	0.853	0.667	1.000	0.927	0.827	0.861
2%	0.933	0.987	**0.900**	0.813	0.973	**0.933**	0.840	**1.000**	**0.987**	0.847	**0.921**
3%	0.920	**1.000**	0.873	**0.833**	0.940	0.907	0.807	0.993	0.947	**0.867**	0.909
4%	**0.940**	0.973	0.873	0.767	0.980	0.920	**0.860**	1.000	0.980	0.787	0.905
6%	0.720	0.980	0.800	0.820	**0.987**	0.900	0.840	0.993	0.947	0.720	0.871
7%	0.887	1.000	0.940	0.673	0.960	0.940	0.593	0.993	0.933	0.847	0.877
16%	0.900	0.933	0.893	0.567	0.960	0.840	0.700	0.987	0.960	0.573	0.831

(b) **RLBench Results**(Success Rate)

Mix Ratio	Rubbish	Umbrella	Laptop	Wine	Frame	Phone	Avg
0%	0.800	0.460	0.180	0.060	0.400	0.100	0.333
5%	0.860	**0.640**	0.200	**0.170**	**0.420**	0.060	**0.392**
10%	0.880	0.580	**0.320**	0.020	0.300	**0.120**	0.370
20%	**0.920**	0.430	0.240	0.060	0.120	0.100	0.312
25%	0.780	0.360	0.120	0.000	0.020	0.060	0.223

Table 4: **Temporal context ablation.** Full history-conditioned trajectories significantly outperform isolated repair segments on LIBERO-Goal.

Trajectory Format	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	Avg
Expert Demonstrations	0.860	0.980	0.860	0.667	0.973	0.853	0.667	**1.000**	0.927	0.827	0.861
Standalone Segments(only repair)	0.900	0.993	0.853	0.673	**0.967**	0.747	0.527	0.973	0.933	0.860	0.843
History-Conditioned(full)	**0.900**	**1.000**	**0.940**	**0.833**	0.960	**0.940**	**0.807**	**1.000**	**0.947**	**0.913**	**0.924**

over five successive iterations of fine-tuning on synthesized repair data, which indicates that the model effectively converts failure experiences into recovery skills.

Mitigation of Dominant Failure Modes. To analyze the qualitative policy evolution, we visualize failure type distributions before and after iterative training in Figure 4. Dominant baseline failure modes are significantly reduced. Concretely, Task 3 mitigates motion planning failures(R5) and object slippage(R4), while Task 6 corrects grasping direction deviations(R2) and gripper orientation errors(R3). This distribution shift validates that synthesized repair data delivers precise learning signals, enabling targeted error correction rather than general policy adjustments.

5 Conclusion

This paper presents RepairVLA, a framework that shifts the paradigm of Vision-Language-Action models from open-loop execution to a robust, closed-loop "**Decompose-Execute-Check-Repair**" architecture. By integrating a VLM-driven monitoring system, the model actively detects failures and dynamically deploys repair

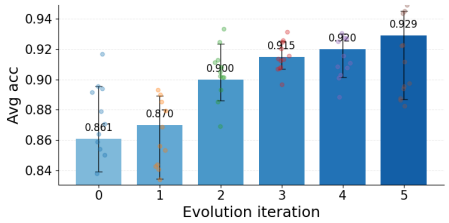


Figure 3: **Evolutionary Progress of Task Success Rate.** The average accuracy across the LIBERO-Goal suite exhibits a steady upward trend over five training iterations. Each bar represents the mean performance, while the scattered individual points indicate results across various data mixture ratios and random seeds, demonstrating the robustness of the improvement.

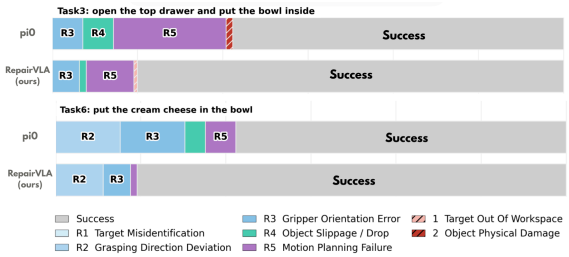


Figure 4: **Distribution of failure modes before and after iterative training.** We evaluate two challenging tasks: *Task 3* (articulated dynamics) and *Task 6* (spatial precision). Inner ring: Success, Recoverable, Irrecoverable. Outer ring: recoverable (R1–R5) and irrecoverable (1–2) error sub-types. RepairVLA notably converts recoverable errors into successes.

strategies to maintain task continuity.

To support this architecture, we introduced an automated **failure-repair data pipeline** that synthesizes high-quality corrective trajectories, successfully bridging the data gap between failure states and valid recovery actions. Empirical results on the RLBench and LIBERO benchmarks confirm that *RepairVLA* significantly enhances policy robustness, achieving a substantial performance gain over current baselines. Furthermore, our iterative training experiments demonstrate a unique capacity for continuous policy evolution, where the agent progressively refines its capabilities by learning from self-generated failure experiences. This work provides a promising foundation for the development of self-improving, adaptive embodied agents in complex task environments.

Limitations

RepairVLA is currently limited to gripper-based robots and tabletop manipulation; extending the repair mechanism to diverse embodiments and real-world scenarios is left as future work.

Although all experiments in this work are conducted in purely simulated environments for academic research without real-world safety hazards, deploying such closed-loop self-repairing robotic systems in physical scenarios may require additional safety verification and human-safety constraints to avoid unexpected manipulation behaviors. Overall, this work offers a promising basis for building self-improving, adaptive embodied agents in complex long-horizon tasks.

Acknowledgments

References

Aishwarya Agrawal, Jiasen Lu, Stanislaw Antol, Margaret Mitchell, C. Lawrence Zitnick, Dhruv Batra, and Devi Parikh. 2016. Vqa: Visual question answering. *Preprint*, arXiv:1505.00468.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, and 26 others. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *Preprint*, arXiv:2204.01691.

Suneel Belkhale, Tianli Ding, Ted Xiao, Pierre Sermanet, Quon Vuong, Jonathan Tompson, Yevgen Chebotar, Debidatta Dwibedi, and Dorsa Sadigh. 2024. Rt-h: Action hierarchies using language. *Preprint*, arXiv:2403.01823.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, and 5 others. 2026. π_0: A vision-language-action flow model for general robot control. *Preprint*, arXiv:2410.24164.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, and 35 others. 2023a. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *Preprint*, arXiv:2307.15818.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, and 32 others. 2023b. Rt-1: Robotics transformer for real-world control at scale. *Preprint*, arXiv:2212.06817.

Hao Chen, Jiaming Liu, Chenyang Gu, Zhuoyang Liu, Renrui Zhang, Xiaoqi Li, Xiao He, Yandong Guo, Chi-Wing Fu, Shanghang Zhang, and Pheng-Ann Heng. 2025. Fast-in-slow: A dual-system foundation model unifying fast manipulation within slow reasoning. *Preprint*, arXiv:2506.01953.

Jaden Clark, Suvir Mirchandani, Dorsa Sadigh, and Suneel Belkhale. 2025. Action-free reasoning for policy generalization. *Preprint*, arXiv:2502.03729.

Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, and 275 others. 2025. Open x-embodiment: Robotic learning datasets and rt-x models. *Preprint*, arXiv:2310.08864.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, and 3 others. 2023. Palm-e: An embodied multimodal language model. *Preprint*, arXiv:2303.03378.

Ran Gong, Xiaohan Zhang, Jinghuan Shang, Maria Vittoria Minniti, Jigarkumar Patel, Valerio Pepe, Riedana Yan, Ahmet Gundogdu, Ivan Kapelyukh, Ali Abbas, Xiaoqiang Yan, Harsh Patel, Laura Herlant, and Karl Schmeckpeper. 2025. Anytask: an automated task and data generation framework for advancing sim-to-real policy learning. *Preprint*, arXiv:2512.17853.

Chenyang Gu, Jiaming Liu, Hao Chen, Runzhong Huang, Qingpo Wuwu, Zhuoyang Liu, Xiaoqi Li, Ying Li, Renrui Zhang, Peng Jia, Pheng-Ann Heng, and Shanghang Zhang. 2025. Manualvla: A unified vla model for chain-of-thought manual generation and robotic manipulation. *Preprint*, arXiv:2512.02013.

Jiayuan Gu, Sean Kirmani, Paul Wohlhart, Yao Lu, Montserrat Gonzalez Arenas, Kanishka Rao, Wenhao Yu, Chuyuan Fu, Keerthana Gopalakrishnan, Zhuo Xu, Priya Sundaresan, Peng Xu, Hao Su, Karol Hausman, Chelsea Finn, Quan Vuong, and Ted Xiao. 2023. Rt-trajectory: Robotic task generalization via hindsight trajectory sketches. *Preprint*, arXiv:2311.01977.

Chi-Pin Huang, Yueh-Hua Wu, Min-Hung Chen, Yu-Chiang Frank Wang, and Fu-En Yang. 2025. Thinkact: Vision-language-action reasoning via reinforced visual latent planning. *Preprint*, arXiv:2507.16815.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tomp-

son, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022. Inner monologue: Embodied reasoning through planning with language models. *Preprint*, arXiv:2207.05608.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, and 17 others. 2025. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *Preprint*, arXiv:2504.16054.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. 2019. Rlbench: The robot learning benchmark & learning environment. *Preprint*, arXiv:1909.12271.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, and 82 others. 2025. Droid: A large-scale in-the-wild robot manipulation dataset. *Preprint*, arXiv:2403.12945.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Paul Foster, Grace Lam, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. 2024. Openvla: An open-source vision-language-action model.

Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. 2017. Dart: Noise injection for robust imitation learning. *Preprint*, arXiv:1703.09327.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as policies: Language model programs for embodied control. *Preprint*, arXiv:2209.07753.

Zefu Lin, Rongxu Cui, Chen Hanning, Xiangyu Wang, Junjia Xu, Xiaojuan Jin, Chen Wenbo, Hui Zhou, Lue Fan, Wenling Li, and Zhaoxiang Zhang. 2025. Embodiedcoder: Parameterized embodied mobile manipulation via modern coding model. *Preprint*, arXiv:2510.06207.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. 2023a. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Preprint*, arXiv:2306.03310.

Zeyi Liu, Arpit Bahety, and Shuran Song. 2023b. Reflect: Summarizing robot experiences for failure explanation and correction. *Preprint*, arXiv:2306.15724.

Weifeng Lu, Minghao Ye, Zewei Ye, Ruihan Tao, Shuo Yang, and Bo Zhao. 2025. Robofac: A comprehensive framework for robotic failure analysis and correction. *Preprint*, arXiv:2505.12224.

Yueen Ma, Zixing Song, Yuzheng Zhuang, Jianye Hao, and Irwin King. 2025. A survey on vision-language-action models for embodied ai. *Preprint*, arXiv:2405.14093.

Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachy Groom, and 37 others. 2025. $\pi^*_{0.6}$: a vla that learns from experience. *Preprint*, arXiv:2511.14759.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. 2022. A generalist agent. *Preprint*, arXiv:2205.06175.

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. *Preprint*, arXiv:1011.0686.

Kohei Sendai, Maxime Alvarez, Tatsuya Matsushima, Yutaka Matsuo, and Yusuke Iwasawa. 2025. Leave no observation behind: Real-time correction for vla action chunks. *Preprint*, arXiv:2509.23224.

Ameesh Shah, William Chen, Adwait Godbole, Federico Mora, Sanjit A. Seshia, and Sergey Levine. 2025. Learning affordances at inference-time for vision-language-action models. *Preprint*, arXiv:2510.19752.

Pengyuan Wang, Fabian Manhardt, Luca Minciullo, Lorenzo Garattoni, Sven Meier, Nassir Navab, and Benjamin Busam. 2021. Demograsp: Few-shot learning for robotic grasping with human demonstration. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5733–5740.

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. 2023. Language to rewards for robotic skill synthesis. *Preprint*, arXiv:2306.08647.

Micha Zawalski, William Chen, Karl Pertsch, Oier Mees, Chelsea Finn, and Sergey Levine. 2025. Robotic control via embodied chain-of-thought reasoning. *Preprint*, arXiv:2407.08693.

Xianchao Zeng, Xinyu Zhou, Youcheng Li, Jiayou Shi, Tianle Li, Liangming Chen, Lei Ren, and Yong-Lu Li. 2025. Diagnose, correct, and learn from manipulation failures via visual symbols. *Preprint*, arXiv:2512.02787.

Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, Ankur Handa, Ming-Yu Liu, Donglai Xiang, Gordon Wetzstein, and Tsung-Yi Lin. 2025. Cot-vla: Visual chain-of-thought reasoning for vision-language-action models. *Preprint*, arXiv:2503.22020.

A Appendix

- **A. Experimental Task Specifications** (Sec. B)
  - RLBench Task Suite (Sec. B)
  - LIBERO-Goal Task Suite (Sec. B)
- **B. VLM Prompt for Failure Diagnosis and Repair Planning** (Sec. C)
- **C. Automated Data Generation Pipeline** (Sec. D)
  - Code-driven Geometric Parameterization (Sec. D)
  - Action Primitive Repair Synthesis (Sec. D)
- **D. Details for Repair Dataset**
  - RLBench Repair Suite (Sec. E.1)
  - LIBERO-Goal Repair Suite (Sec. E.2)
- **E. Visualization of Autonomous Repair Performance** (Sec. F)
- **F. Iterative Training: Fixed-Ratio Replay vs. Cumulative Data Expansion** (Sec. G)

B Experimental Task Specifications

To comprehensively evaluate the failure diagnosis and recovery capabilities of *RepairVLA*, we select tasks from two distinct benchmarks: RLBench and LIBERO-Goal. These tasks are chosen to cover a wide spectrum of physical constraints, including articulated dynamics, narrow passage constraints, and high-precision alignment.

B.1 RLBench Task Suite

We select six tasks from RLBench that represent failure modes requiring physical reasoning rather than simple re-execution. Table 5 details the descriptions of each task.

B.2 LIBERO-Goal Task Suite

We also utilize the **LIBERO-Goal** suite from the LIBERO benchmark to evaluate behavioral adaptation and failure recovery across varying objectives within a fixed environmental layout, as shown in Figure 5. This suite is specifically selected to isolate the transfer of procedural knowledge from declarative features. Table 6 details the 10 tasks used in our evaluation, ordered from task0 to task9.

Figure 5: The constant initial scene layout shared across the 10 tasks in the LIBERO-Goal suite.

C VLM Prompt for Failure Diagnosis and Repair Planning

As illustrated in the Dual-Stream Inference Architecture, the VLM serves as a high-level reasoner. We design a structured prompt to guide the VLM in performing three objectives: (1) **Subtask Decomposition**, decomposes the video into semantic sub-tasks and identifies the moment of failure; (2) **Failure Diagnosis**, detecting anomalies based on physical constraints; and (3) **Repair Planning**, synthesizing a sequence of corrective plans to recover from the failure.

To ensure the generated repair plans can be translated into executable Action Primitives (as shown in the Automated Data Generation Pipeline), we enforce a JSON output schema, which is presented in **Listing 1**. The prompt dynamically incorporates task-specific contexts (e.g., scene descriptions and success criteria). Below, we present the prompt template instantiated for the "Close Laptop Lid" task.

D Automated Data Generation Pipeline

To bridge the gap between high-level reasoning and low-level control, we adopt a neuro-symbolic strategy to solve the data scarcity problem in robot repair scenarios. Specifically, the pipeline can be decomposed into two code generation stages: **Ge-**

Task Name	Instruction & Description	Select Reason	Rendering
Close Laptop Lid	Close the laptop lid completely by pushing or grasping the edge.	**Articulated Dynamics**: Requires reasoning over a rotational hinge axis. Failures typically involve screen collisions or incorrect force vectors that deviate from the hinge axis, leading to slippage or misalignment.	
Phone on Base	Grasp the phone and precisely place it onto its charging base.	**High-Precision Alignment**: Demands centimeter accuracy. The primary challenge lies in the spatial precision required to seat the phone into the corresponding base socket.	
Put Rubbish in Bin	Pick up the rubbish and leave it inside the designated trash can.	**Multi-Stage Transition**: Evaluates object selection and grasp stability. Failures often involve grasping non-target objects or adopting improper grasp poses for thin, geometrically complex debris.	
Stack Wine	Slide the wine bottle into the narrow rack without collisions.	**Narrow Passage Constraints**: The bottle must enter a tight slot. Slight deviations lead to immediate contact with the rack frame.	
Take Umbrella Out	Grasp the umbrella handle and lift it vertically out of the stand.	**Vertical Clearance**: Misalignment during the lift causes the handle to jam within the stand or inadvertently knock the stand over.	
Take Frame Off	Slide the photo frame off the hanger and set it down on the table.	**Contact-Rich Interaction**: Involves delicate direction management. Failures often stem from the frame catching on the hook during the slide.	

Table 5: Detailed specifications of selected RLBench tasks. Each task is chosen to expose specific physical and geometric constraints.

ometric Parameterization and **Action Primitive Synthesis**.

D.1 Stage 1: Code-driven Geometric Parameterization

In the first stage, the VLM analyzes segmented semantic point clouds and key frames to extract the physical and functional parameters required for manipulation (detailed in Listing 2). For the "Close Laptop Lid" task, the model must not only identify the object's geometry but also infer the kinematic constraints of the hinge axis.

D.2 Stage 2: Action Primitive Repair Synthesis

The second stage maps the identified geometric parameters to a sequence of Action Primitives (as illustrated in Listing 3). This ensures the generated repair trajectory is physically feasible and respects the identified environmental constraints, such as the rotational limits of the articulated joint.

E Details for Repair Dataset

This section provides technical details for the repair dataset used to fine-tune the RepairVLA model, including the specific data modalities, collection scales, and formatting for the RLBench and LIBERO-Goal benchmarks.

E.1 RLBench Repair Suite

We construct a targeted repair dataset using the automated pipeline described in our methodology. For each of the six selected RLBench tasks, we collect 100 successful repair trajectories. To ensure reproducibility, each trajectory is initialized with a unique random seed and the full execution process is recorded. These trajectories capture a transition from a diverse set of failure states to a successful task completion through synthesized Action Primitives.

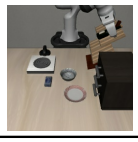
ID	Instruction & Description	Select Reason	Rendering
task0	Open the middle drawer of the cabinet.	**Articulated Dynamics**: Requires identifying the prismatic joint axis. Failures often occur if the force vector deviates from the sliding axis.	
task1	Put the bowl on the stove.	**Surface Alignment**: Demands precise placement on a target burner while avoiding collisions with neighboring kitchen appliances.	
task2	Put the wine bottle on top of the cabinet.	**High-Reach Manipulation**: Involves lifting a slender object to an elevated surface, testing vertical clearance reasoning and release stability.	
task3	Open the top drawer and put the bowl inside.	**Sequential Precision**: Combines articulated opening with constrained placement, evaluating the model's ability to sequence multi-stage corrective motions.	
task4	Put the bowl on top of the cabinet.	**Goal-Oriented Generalization**: Verifies if the repair logic for high-reach tasks generalizes across different object geometries (bottle vs. bowl).	
task5	Push the plate to the front of the stove.	**Non-Prehensile Interaction**: Evaluates planar sliding where maintaining constant vertical contact force is critical for controlling the plate's trajectory.	
task6	Put the cream cheese in the bowl.	**Relative Positioning**: Requires sub-centimeter accuracy to seat a small box inside a bowl without colliding with its rim.	
task7	Turn on the stove.	**Contact-Rich Interaction**: Demands precise torque application on a small-scale knob, where slipping is a common failure mode.	
task8	Put the bowl on the plate.	**Stability Control**: Evaluates stable stacking; minor base misalignment frequently leads to the bowl tipping off the plate surface.	
task9	Put the wine bottle on the rack.	**Narrow Passage**: The bottle must enter a tight slot, requiring high-accuracy orientation and lateral positioning to avoid rack frame contact.	

Table 6: Detailed specifications of the LIBERO-Goal suite. All tasks are performed in the same environment to evaluate instruction-driven behavioral adaptation.

```
# Prompt template aligning with the RepairVLA Framework
user_prompt_text = (
    f"**Task Instruction:** '{task_instruction}'\n"
    f"**Total Video Length:** {total_frames} frames.\n\n"

    "**Scene Context:** An open laptop is on the table. The robot must push or grasp the laptop lid to close it completely.\n\n"

    "**INSTRUCTIONS:**\n"
    "You act as a failure analyst and motion planner. Process the video step-by-step:\n"
    "1. **Segmentation:** Identify the start and end frames for standard sub-tasks: 'Approach', 'Interact', and 'Close'.\n"
    "2. **Status Check:** Validate the success of each sub-task against the visual evidence.\n"
    "3. **Diagnosis:** If a failure occurs (e.g., collision, slippage, incomplete closure), pinpoint the specific 'Failure End'
      frame and explain the physical reason.\n"
    "4. **Repair Planning:** Generate a step-by-step recovery plan. The plan must consist of high-level atomic actions (e.g.,
      'Open Gripper', 'Lift Up', 'Re-Approach') that correct the identified error.\n\n"

    "**CRITICAL SUCCESS CRITERIA:**\n"
    "- **Close Success:** The lid must be **completely flat** against the base. If there is a visible gap or the lid springs back
      up, set `success: false`.\n"
    "- **Safety:** If the robot smashes the screen or pushes the whole laptop off the table, set `success: false`.\n\n"

    "**REQUIRED OUTPUT (JSON):**\n"
    "Return a single JSON object. If a failure is detected, the 'repair_plan' list is mandatory.\n"
    "Refer to the following **EXAMPLE FORMAT** (do not copy the values, generate based on the video):\n"
    "{\n"
    "  \"subtasks\": [\n"
    "    {\"start\": 0, \"end\": 45, \"type\": \"Approach\", \"success\": true},\n"
    "    {\"start\": 45, \"end\": 80, \"type\": \"Interact\", \"success\": false}\n"
    "  ],\n"
    "  \"failure_analysis\": {\n"
    "    \"is_failure\": true,\n"
    "    \"failure_frame\": 80,\n"
    "    \"reason\": \"The gripper approached too low and collided with the screen edge, preventing a proper grasp.\"\n"
    "  },\n"
    "  \"repair_plan\": [\n"
    "    \"Open Gripper to release tension\",\n"
    "    \"Lift up vertically by 10cm to clear the obstacle\",\n"
    "    \"Re-orient gripper to align parallel with the lid\",\n"
    "    \"Approach the lid center slowly\"\n"
    "  ]\n"
    "}"
)
```

Listing 1: Instantiated VLM Prompt for Analysis and Repair Generation. The JSON schema enforces structural output for downstream parsing.

E.1.1 Data Format and Modalities

The dataset is stored in a structured sequence format. Each transition within a rollout segment contains visual observations and proprioception. The specific fields recorded at each timestep include:

- **Visual Observations**: RGB images (256×256) from the `front_rgb` and `wrist_rgb` views.
- **Proprioceptive State**: Includes `joint_positions` (7-DOF), `gripper_pose` (7-DOF), and the `gripper_open` state.
- **Control Signals**: The action field stores the target pose commands, serving as the ground-truth for behavioral cloning.
- **Task Metadata**: A reward signal and a terminate flag for trajectory termination.

E.1.2 Dataset Statistics

In total, the RLBench repair suite comprises 600 high-quality trajectories. This provides a dense supervisory signal for learning recovery behaviors that are typically absent in standard datasets.

E.2 LIBERO-Goal Repair Suite

We also construct a repair dataset on the libero_goal suite using the automated pipeline described in our methodology. Unlike general-purpose demonstration corpora, this dataset is explicitly designed to capture *recovery behaviors*: trajectories that start from diverse failure configurations and proceed to successful task completion through iterative correction.

For each LIBERO-Goal task, we first collect failures under our deployed policy across multiple runs. We then generate repair trajectories by restoring the environment to a recorded failure state and executing a corrected action sequence. To support continual improvement, the dataset

```python
def extract_laptop_geometry(lid_pcd, base_pcd):
    """
    Code 1: Geometric Parameterization to extract physical attributes.
    Refers to the 'Physical Parameters' and 'Generating Param' modules in Figure 2.
    """
    # 1. Denoising and segmentation of articulated parts
    # Informed by the EmbodiedCoder paradigm for geometric fitting
    lid_main = dbscan_denoise(lid_pcd)
    base_main = dbscan_denoise(base_pcd)

    # 2. Fit the lid to a rectangle to get surface normal and center
    # Corresponds to the 'Physical Parameters' box in Figure 2
    lid_ctr, lid_norm, _, _ = fit_plane_pca(lid_main)
    lid_geom = project_to_3d_rectangle(lid_main, lid_ctr, lid_norm)

    # 3. Infer the functional hinge axis relative to the base
    # This geometric reasoning is essential for rotational articulated tasks
    hinge_axis = infer_hinge_line(lid_geom, base_main)

    # Calculate the opening angle for trajectory planning
    initial_angle = calculate_angle(lid_norm, [0, 0, 1])

    return {
        "lid_center": lid_ctr,
        "hinge_axis": hinge_axis,
        "current_angle": initial_angle
    }
```

Listing 2: Code 1: Geometric Parameterization for extracting physical attributes and functional constraints. This refers to the 'Physical Parameters' and 'Generating Param' modules in the framework overview.

aggregates failures and repairs collected across *multiple training stages* (e.g., early/late checkpoints and iterative re-training rounds). Therefore, the final corpus represents a union of heterogeneous failure modes observed throughout evolution, rather than a single snapshot from one fixed policy.

Importantly, for every task and every failure type we consider, we collect **more than 10** repair trajectories, ensuring sufficient coverage for learning robust recovery strategies and enabling subsequent iterations of repair-policy training.

E.2.1 Data Format and Modalities

The dataset is stored in a structured, step-wise sequence format compatible with DexBotic training pipeline. Each transition is written as one JSON line and contains:

- **Language Instruction**: `prompt`, the natural-language goal instruction for the current task instance.
- **Visual Observations**: RGB images(256×256) from both agent view and wrist view . Each view is stored as an image file and referenced in the JSONL entry, preserving view identity without concatenating streams.
- **Proprioceptive State**: The low-dimensional robot state vector used by the policy (e.g., end-effector pose and gripper state), recorded per step for precise temporal alignment with observations.
- **Recoverable Simulator Snapshot**: `recover`, a path to a compressed state blob that deterministically restores the simulator to the exact scene configuration at that timestep.
- **Step and Episode Metadata**: timestep index, episode id, and outcome indicators (e.g., `success`) for termination and filtering.

E.2.2 Dataset Statistics and Composition

The LIBERO-Goal repair suite is organized by task and failure type. Because data are collected throughout iterative training, the suite contains a mixture of failure distributions from different model stages. This staged aggregation yields a broader coverage of corner cases, and provides a dense supervisory signal for recovery behaviors that are typically absent in standard task-success datasets.

F Visualization of Autonomous Repair Performance

This section qualitatively demonstrates the capability of **RepairVLA** to diagnose and recover from failure modes through a series of

```python
def synthesize_laptop_repair(params, failure_diagnosis):
    """
    Code 2: Synthesizing trajectories from parameterized geometry.
    Refers to the 'Action Primitive' and 'Repair Trajectory' synthesis in Figure 2.
    """
    repair_trajectory = []
    hinge = params["hinge_axis"]

    # Logic: Address failure (e.g., 'Grasping Deviation') using precise Action Primitives
    # Step 1: Safety maneuver - clear the failure point
    # Reference: Figure 2 'Post-Failure Actions' segment
    repair_trajectory.append(ActionPrimitive.LIFT_UP(height=0.15))

    # Step 2: Spatial re-alignment using inferred hinge and lid geometry
    # The target pose is adjusted relative to the lid's center and hinge axis
    alignment_pose = calculate_alignment(params["lid_center"], hinge)
    repair_trajectory.append(ActionPrimitive.APPROACH(alignment_pose))

    # Step 3: Rotational interaction synthesized along the identified hinge axis
    # The closing motion is parameterized by the inferred rotation radius and center
    repair_trajectory.append(ActionPrimitive.CLOSE_MOTION(
        rotation_center=hinge.origin,
        rotation_axis=hinge.direction,
        target_angle=0.0
    ))

    return repair_trajectory
```

Listing 3: Code 2: Mapping repair plans to executable Action Primitives. This refers to the 'Action Primitive' and 'Repair Trajectory' synthesis in the framework overview.

keyframe visualizations. As illustrated in Figure 6, we present representative trajectories from both the LIBERO-Goal and RLBench benchmarks to demonstrate the model's capability to diagnose failure states and execute corrective actions.

After finetuning without the targeted repair data, the base policy often exhibits limited self-correction once it enters a failure state, repeatedly applying locally consistent but task-invalid motions. Figure 7 presents qualitative evidence on two LIBERO-Goal tasks: Task4 (*Put the bowl on top of the cabinet*) and Task9 (*Put the wine bottle on the rack*). For each task, we align (i) a representative test-time failure rollout of *pi0*, (ii) the corresponding repair demonstrations collected for that failure type, and (iii) the test-time behavior of **RepairVLA** after finetuning on these repairs, showing that the finetuned model executes corrected alignment and placement to complete the task.

G Iterative Training: Fixed-Ratio Replay vs. Cumulative Data Expansion

In our iterative experiments on LIBERO-GOAL, the raw dataset is fixed to the original demonstration set, denoted as $\mathcal{D}_{\text{raw}}$. Let $\mathcal{R}^{(k)}$ be the set of failure-repair trajectories synthesized from failures collected after iteration k. We consider two practical dataset update settings:

(1) Fixed-ratio replay (constant-size mixing). At iteration $k{+}1$, we mix a fixed fraction ρ of the *previous-stage* repair set into $\mathcal{D}_{\text{raw}}$:

$$\mathcal{D}^{(k+1)} = \mathcal{D}_{\text{raw}} \cup \text{Sample}\Big(\mathcal{R}^{(k)}, \rho\Big), \quad (1)$$

where $\text{Sample}(\cdot, \rho)$ subsamples repair trajectories to maintain an approximately constant mixture ratio.

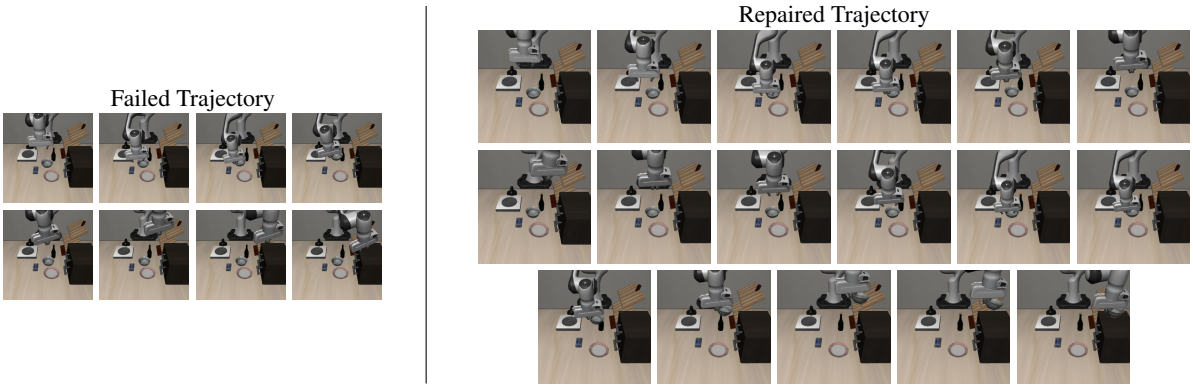
(2) Cumulative expansion (dataset aggregation). At iteration $k{+}1$, we keep *all* previously collected repair trajectories:

$$\mathcal{D}^{(k+1)} = \mathcal{D}^{(k)} \cup \mathcal{R}^{(k)} = \mathcal{D}_{\text{raw}} \cup \bigcup_{i=0}^{k} \mathcal{R}^{(i)}. \quad (2)$$

Algorithm 3 Iterative training with two dataset update settings

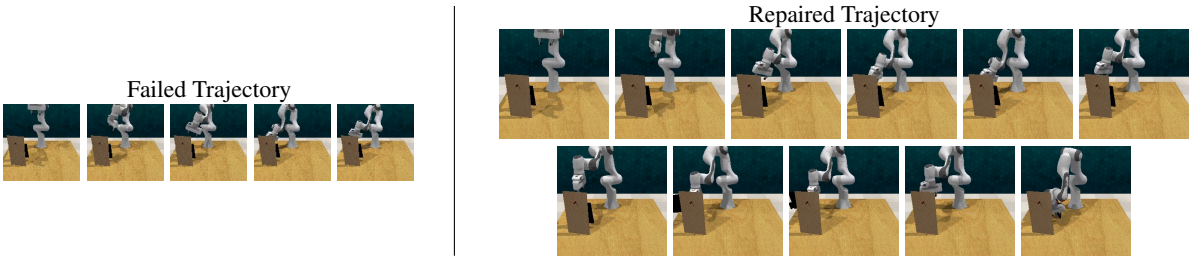
Require: Raw dataset $\mathcal{D}_{\text{raw}}$, initial policy $\pi^{(0)}$, iterations K, mix ratio ρ
$\mathcal{D}^{(0)} \leftarrow \mathcal{D}_{\text{raw}}$
for $k = 0$ **to** $K - 1$ **do**
  Roll out $\pi^{(k)}$ on LIBERO-GOAL to collect failures; synthesize repair set $\mathcal{R}^{(k)}$
  if Fixed-ratio replay **then**
    $\mathcal{D}^{(k+1)} \leftarrow \mathcal{D}_{\text{raw}} \cup \text{Sample}(\mathcal{R}^{(k)}, \rho)$
  else if Cumulative expansion **then**
    $\mathcal{D}^{(k+1)} \leftarrow \mathcal{D}^{(k)} \cup \mathcal{R}^{(k)}$
  end if
  Fine-tune to obtain $\pi^{(k+1)}$ on $\mathcal{D}^{(k+1)}$ (11 epochs per stage in our setting)
end for

LIBERO-Goal (Task: *Put the bowl on the top of the cabinet*)



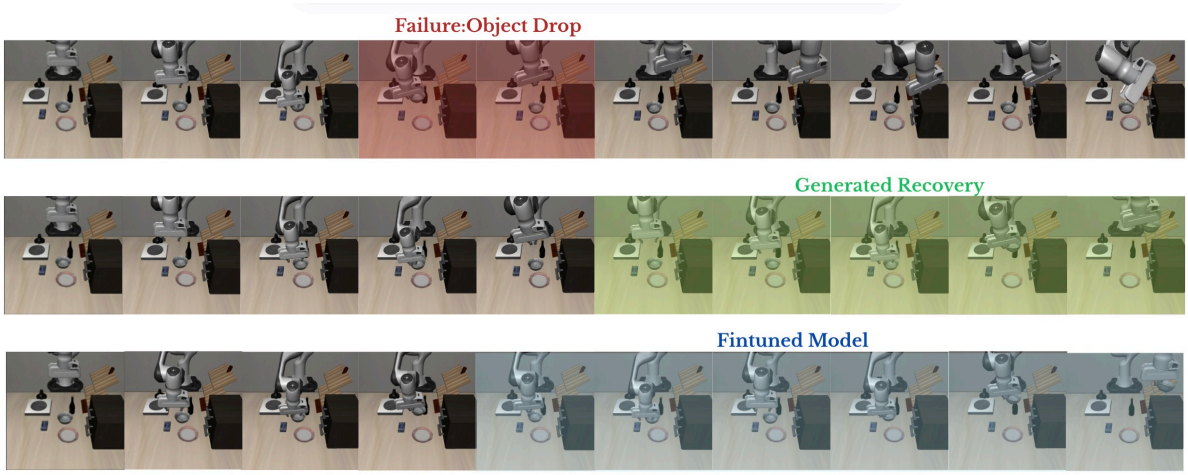
(a) Qualitative comparison on LIBERO-Goal. The failed sequence (left) illustrates a failure to grasp the bowl securely. *RepairVLA* (right) diagnoses the unstable grasp and executes an autonomous re-grasping maneuver before successfully placing the bowl on the target cabinet.

RLBench (Task: *Take Frame Off Hanger*)

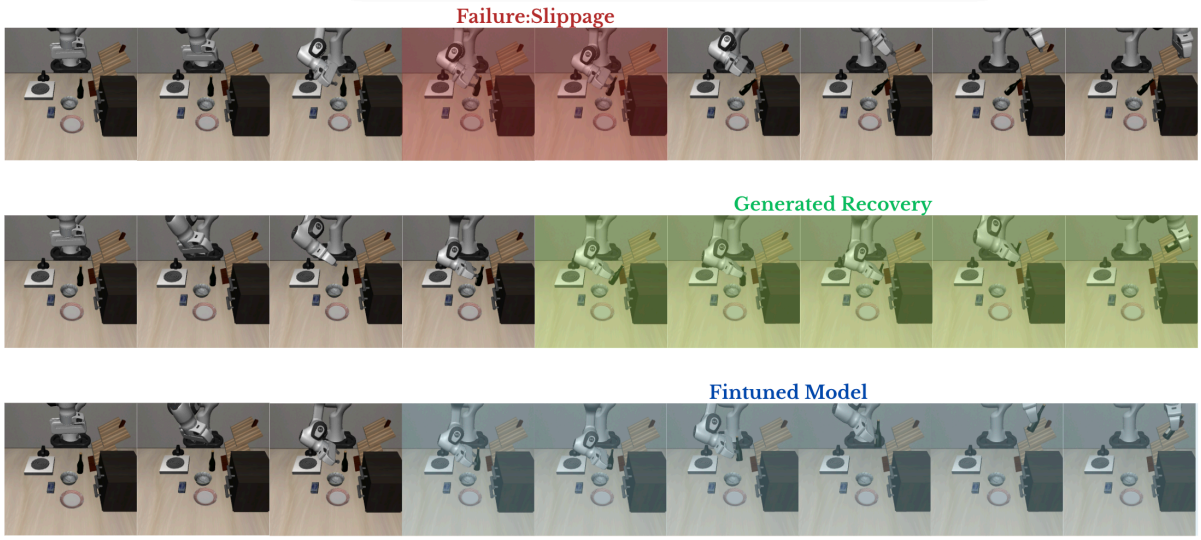


(b) Qualitative comparison on RLBench. The base policy (left) attempts to extract the frame in an incorrect direction, causing it to catch on the hook. *RepairVLA* (right) corrects this by executing a vertical lifting primitive to slide the frame off the hanger smoothly.

Figure 6: Qualitative keyframe visualization of failure and recovery trajectories across the LIBERO-Goal (top) and RLBench (bottom) suites.



(a) **Task4**: *Put the bowl on top of the cabinet.*



(b) **Task9**: *Put the wine bottle on the rack.*

Figure 7: **Qualitative alignment of failure, repair data, and post-finetuning behavior.** For each task, **top**: test-time failure rollout of the base *pi0* policy; **middle**: repair demonstrations collected for the same failure type; **bottom** (*RepairVLA*): test-time rollout after finetuning on the corresponding repair data, showing successful recovery and task completion.

Sampling rule. $\mathrm{Sample}(\mathcal{R}^{(k)}, \rho)$ denotes uniform sampling at the episode level such that the number of sampled repair episodes is approximately $\rho|\mathcal{D}_{\text{raw}}|$ (measured by episodes).

This setting continuously enlarges the training corpus and empirically yields stronger performance in our main evolution results.

Table 7 reports two instantiations of **fixed-ratio replay** under an identical two-stage schedule (11 epochs per stage). Both settings underperform cumulative expansion, which is expected: retaining all past repair data better preserves diverse failure distributions and mitigates forgetting under distribution shift, consistent with dataset aggregation and rehearsal-style arguments in sequential decision learning.

Table 7: **Fixed-ratio replay across iterations on LIBERO-GOAL.** Two-stage iterative training (11 epochs per stage). "2%→4%" mixes 2% repair data in stage-1 and 4% newly collected repair data in stage-2, with $\mathcal{D}_{\text{raw}}$ fixed.

Setting	T0	T1	T2	T3	T4	T5
2%→4%	0.8867	1.0000	0.8933	0.6400	0.9800	0.9933
2%→2%	0.9933	0.9867	0.9133	0.7267	0.9733	0.9533

Setting	T6	T7	T8	T9	Avg
2%→4%	0.7933	0.9867	0.9400	0.9200	0.9033
2%→2%	0.7400	0.9733	0.9800	0.7600	0.9000

Why cumulative expansion outperforms fixed-ratio replay. Fixed-ratio replay keeps only a small subset of past repair trajectories, while the failure-state distribution can drift as the policy evolves, leading to forgetting of early corner cases under covariate shift. In contrast, cumulative expansion is closer to dataset aggregation: retaining historical repairs covers a broader support of failure states and typically yields more stable, monotonic improvements, at the cost of a growing dataset and longer training. This intuition aligns with classic imitation-learning analyses of distribution shift and dataset aggregation (e.g., DAgger) as well as rehearsal-style continual learning methods that mitigate forgetting via episodic memory or replay.