POLICYGROUND: RECOVERING FROM HIGH-LEVEL FAILURES BY PROBING FROZEN HIERARCHICAL VISION-LANGUAGE-ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Hierarchical vision-language-action (VLA) systems support long-horizon manipulation by decomposing a task into language subtasks executed by a low-level policy. After a high-level failure, however, a semantically correct recovery instruction may still be unsupported by the particular policy that must execute it. Assessing this support is difficult for multi-step recovery: later policy responses depend on states produced by earlier actions, so policy support cannot generally be assessed from a single policy query, while physically trying each candidate defeats one-shot recovery. We introduce POLICYGROUND, a training-free framework for recovering from high-level failures in frozen, black-box hierarchical VLAs. POLICYGROUND first derives a policy-independent recovery goal from the recorded failed episode. Its key mechanism, Chain-of-Probe, then connects high- and low-level target-policy responses through proxy contexts retrieved from the failed episode, constructing a proxy multi-step response before physical execution. This policy-specific evidence is used to evaluate and revise the recovery instruction toward one supported by the target policy. We evaluate POLICYGROUND with RoboMME-FR, a one-shot physical recovery protocol over ten history-dependent RoboMME tasks. Across these tasks, POLICYGROUND improves micro-averaged recovery success from 7.7% to 23.4%.

1 INTRODUCTION

Vision-language-action (VLA) models map visual observations and language instructions to robot actions, supporting a broad range of manipulation tasks (Zitkovich et al., 2023; Kim et al., 2025; NVIDIA et al., 2025; Team et al., 2025; Intelligence et al., 2026). Hierarchical systems extend VLAs to longer horizons by using a high-level policy to generate language subtasks and a low-level policy to execute them (Shi et al., 2025; Sridhar et al., 2026; Torne et al., 2026). Despite these advances, high-level policies can lose track of task progress, overlook earlier observations, or select the wrong object. Once an incorrect subtask changes the environment, simply resuming the original plan may no longer satisfy the task.

A general-purpose large multimodal model (LMM) can inspect the failed episode and propose a semantically plausible correction without task-specific recovery training. Semantic correctness, however, does not imply that the target VLA can realize the resulting instruction. The supervisor and target policy may use different action abstractions, object references, or task vocabularies, and the failure state may introduce policy-dependent prerequisites absent from the original plan. For example, after the robot lifts the wrong container, an instruction to pick up the correct container identifies the desired outcome but may cause the policy to keep lifting the already-held object unless it is first instructed to release it. Thus, recovery requires not only determining what remains to be achieved, but also finding an instruction through which the particular frozen policy can achieve it.

The central technical challenge is that such policy support is inherently closed-loop for multi-step recovery. Whether an instruction is supported by the target policy cannot generally be determined from its text or from a single target-policy query: later high-level subtasks and low-level actions depend on states produced by earlier actions. Physically executing every candidate would reveal these responses, but is incompatible with settings in which only one recovery attempt should be

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.

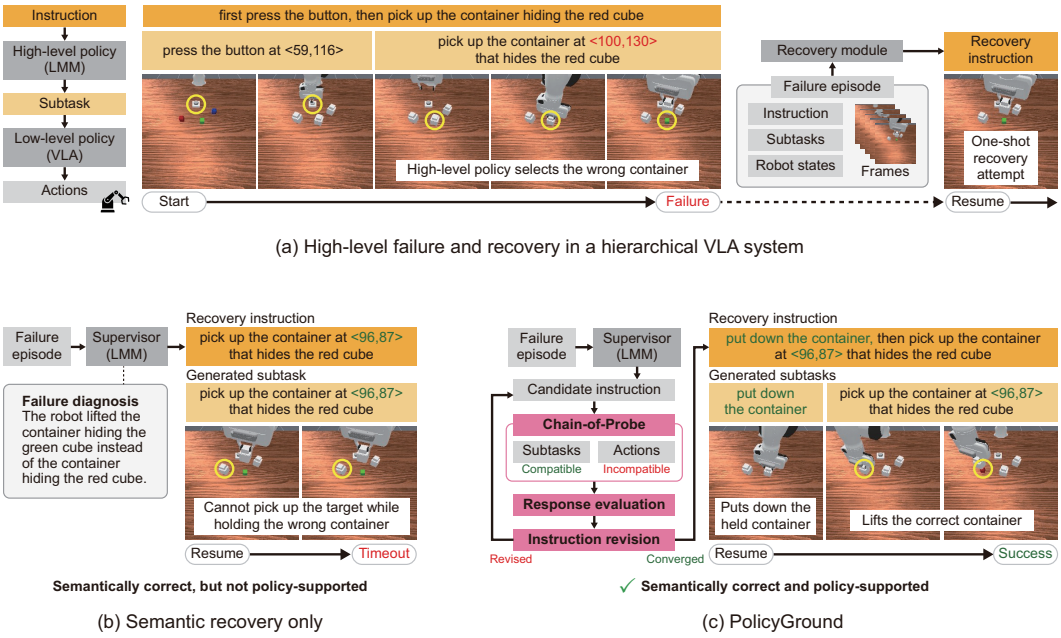


Figure 1: Illustration of the distinction between semantic and policy-supported recoverability. (a) An incorrect high-level subtask causes the robot to lift the wrong container. The recovery module receives the recorded failure episode and restarts the context-reset policy from the same physical failure state with a recovery instruction. (b) Under semantic recovery only, the supervisor correctly diagnoses the failure and identifies the target container, but its instruction omits the prerequisite of releasing the currently held container. The instruction is semantically correct but not supported by the target policy, leading to a timeout. (c) POLICYGROUND probes the frozen hierarchy before physical execution, detects this incompatibility, and revises the instruction to first put down the held container and then pick up the target container. The revised instruction is both semantically correct and policy-supported, enabling successful one-shot physical recovery.

committed to the environment. The challenge is therefore to obtain policy-specific evidence about a multi-step recovery instruction before physically executing it.

Existing recovery methods address complementary parts of this problem. Failure-specialized models diagnose failures or generate corrective guidance from failure data (Duan et al., 2025; Lu et al., 2025; Zeng et al., 2026; Lin et al., 2026), while planning and runtime recovery systems use language models, structured planning, or feedback observed during execution to revise robot behavior (Liu et al., 2023; Zhi et al., 2025; Guo et al., 2026). These approaches can recover from observed failures, but do not directly evaluate the multi-step response of the same frozen, black-box hierarchical policy to a candidate high-level recovery instruction before committing it to the environment.

We introduce POLICYGROUND, a training-free framework that grounds recovery instructions in the observed behavior of a frozen, black-box hierarchical VLA (Figure 1). We distinguish semantic recoverability, which concerns identifying a recovery outcome consistent with the original task and failed episode, from policy-supported recoverability, which concerns whether the target policy can realize that outcome from the failure state. Semantic Recovery Specification addresses the former by deriving a policy-independent recovery goal from the recorded failed episode. Policy-Grounded Realization addresses the latter by probing the target policy and revising how that goal is expressed.

The key mechanism is Chain-of-Probe. A direct multi-step probe would require future observations that do not exist until the candidate is physically executed. Instead of predicting those observations with a learned world model, Chain-of-Probe reuses contexts recorded in the failed episode as proxies. It alternates high-level subtask prediction and low-level action probing, retrieves a proxy context compatible with the predicted action outcome, and uses that context to continue the policy response. In this way, it stitches black-box high- and low-level responses into a proxy multi-step interaction.

A frozen supervisor evaluates the resulting policy-specific evidence and revises the candidate instruction when its semantic intent and the target policy's response disagree. Only the selected final instruction is physically executed.

We evaluate POLICYGROUND using RoboMME (Dai et al., 2026), whose history-dependent tasks expose high-level errors that cannot always be diagnosed from the current observation alone. We construct RoboMME-FR, a one-shot recovery protocol that evaluates competing methods from identical saved failure states across ten tasks. POLICYGROUND improves average recovery success from 7.7% to 23.4% and increases end-to-end task success from 49.6% to 58.2% relative to semantic recovery. Component ablations further show distinct roles for high- and low-level policy feedback in recovery.

Our contributions are:

- We formulate high-level failure recovery as requiring both semantic recoverability and policy-supported recoverability, and identify a key obstacle to the latter: multi-step policy support is a closed-loop property that cannot generally be assessed from an instruction or a single policy query.
- We propose POLICYGROUND and its Chain-of-Probe mechanism, which connects black-box high- and low-level policy responses through proxy contexts retrieved from the recorded failed episode. This enables multi-step recovery candidates to be evaluated and revised using target-policy-specific evidence before physical execution.
- We introduce RoboMME-FR, a one-shot physical recovery protocol over ten history-dependent tasks, and analyze recovery performance, hierarchy-specific policy feedback, revision depth, and inference cost.

2 RELATED WORK

Hierarchical VLAs and Memory for Long-Horizon Manipulation. Vision-language-action (VLA) models map visual observations and language instructions to robot actions, while hierarchical systems extend this paradigm by separating high-level task reasoning from low-level action execution (Zitkovich et al., 2023; Kim et al., 2025; Black et al., 2025b;a; Sridhar et al., 2026; Torne et al., 2026; Wang et al., 2026). Recent work further studies memory for long-horizon and non-Markovian manipulation. MemER retrieves task-relevant observations to condition high-level subtask generation, while MEM combines complementary textual and visual memory (Sridhar et al., 2026; Torne et al., 2026). Recent benchmarks evaluate manipulation tasks whose correct decisions depend on information accumulated over time (Dai et al., 2026; Chung et al., 2026; Lei et al., 2026). These works primarily study whether memory enables successful nominal execution. Our setting instead begins after an incorrect high-level subtask has already changed the environment. We retain the recorded failed episode externally for failure diagnosis and proxy-context retrieval, rather than restoring it as pre-failure target-policy history.

Failure Detection and Learned Recovery. A growing body of work learns failure detection, explanation, or corrective behavior from failure-specific data. AHA trains a vision-language model to reason about execution failures, RoboFAC learns failure understanding and corrective guidance from failed trajectories, and ViFailback produces textual and visual corrections for failed VLA executions (Duan et al., 2025; Lu et al., 2025; Zeng et al., 2026). FailSafe further learns corrective low-level actions from synthesized failure data (Lin et al., 2026). Related systems such as RACER use an external vision-language supervisor to provide corrective feedback to a learned visuomotor policy (Dai et al., 2025). RePO-VLA instead incorporates success, failure, and recovery trajectories directly into policy optimization to learn recovery-aware behavior (Liufu et al., 2026). These approaches demonstrate the value of failure-specific supervision, but typically require additional failure data, model training, or a specialized recovery model. In contrast, we assume that a failure signal is available and focus on a different question: how to realize a high-level recovery goal through an already frozen, black-box hierarchical VLA. Failure detection itself is therefore complementary to POLICYGROUND.

Planning and Runtime Recovery. Several methods support recovery without directly retraining the underlying robot policy. REFLECT summarizes failed experience and uses an LLM to ex-

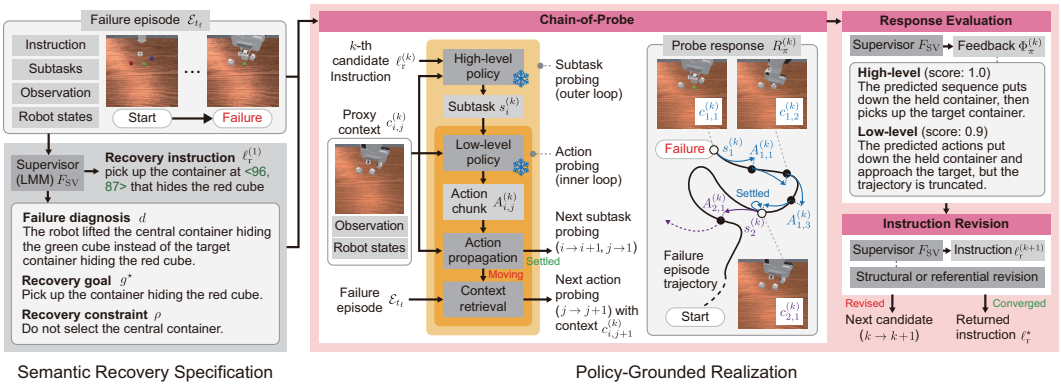


Figure 2: Overview of POLICYGROUND. Semantic Recovery Specification derives a failure diagnosis, recovery goal, semantic constraint, and initial recovery instruction from the recorded failed episode. For each candidate, Chain-of-Probe alternates frozen high-level subtask prediction and low-level action probing, connecting successive policy responses through proxy contexts retrieved from the episode without physical execution. The supervisor evaluates the hierarchy-specific responses and revises the best-so-far instruction until convergence or the revision budget is reached; only the returned instruction is physically executed. Snowflakes denote frozen target-policy components.

plain failures and revise subsequent plans (Liu et al., 2023). Recover uses structured scene and task knowledge for recovery planning, while RePLan revises high- and low-level plans in response to observed execution outcomes (Cornelio & Diab, 2024; Skreta et al., 2024). DoReMi couples planning and execution through runtime constraints, while more recent systems use VLM-based feasibility or symbolic-effect verification to trigger replanning during execution (Guo et al., 2023; Zhi et al., 2025; Guo et al., 2026). Recent systems also study proactive self-correction by predicting impending failures and backtracking to earlier subtasks (Ma et al., 2026). Other work optimizes prompts for black-box VLM-based control and recovery (Chen et al., 2025). These approaches use external reasoning models, structured state representations, or feedback obtained from actual execution to revise robot behavior. In contrast, POLICYGROUND asks whether a candidate high-level recovery instruction is supported by the same frozen hierarchical policy that will execute it, before committing the instruction to the environment. This requires reasoning about multi-step policy responses whose later inputs depend on earlier actions. Chain-of-Probe addresses this by connecting black-box high- and low-level policy responses through proxy contexts retrieved from the recorded failed episode, providing policy-specific evidence before physical execution. Recent policy-aware recovery methods operate at different interfaces: ProbeAct uses an auxiliary probe over internal VLA representations to correct low-level actions (Zhang et al., 2026), while B2FF uses precomputed visual milestones to guide recovery (Shin et al., 2026). Unlike these methods, POLICYGROUND requires neither internal policy representations nor a predefined visual recovery trajectory, and instead uses black-box policy responses to evaluate and revise high-level natural-language recovery instructions. Chain-of-Probe has also been used to probe confidence across LLM chain-of-thought steps (Wang et al., 2025); our use instead refers to chaining hierarchical VLA responses through retrieved proxy contexts.

3 METHOD

3.1 PROBLEM FORMULATION

We consider a frozen hierarchical vision-language-action system $\pi = (\pi_{\mathrm{H}}, \pi_{\mathrm{L}})$. At decision step i, let o_i denote the current visual observation, q_i the robot proprioceptive state (i.e. joint angles and gripper state), and h_i the execution history available to the high-level policy, including previously observed states and issued subtasks. Given a task instruction ℓ, the high-level policy generates a language subtask s_i, which the low-level policy maps to an action chunk A_i:

$$s_i \sim \pi_{\mathrm{H}}(\ell, h_i, o_i), \qquad A_i \sim \pi_{\mathrm{L}}(s_i, o_i, q_i). \tag{1}$$

We record the resulting episode as $\mathcal{E}_t = \{(o_\tau, q_\tau, s_\tau)\}_{\tau=1}^{t}$, from which h_i is constructed according to the target high-level policy. When a failure is detected at time t_f, recovery preserves the physical state but resets the target-policy context. The recovery module retains $\mathcal{E}_{t_\mathrm{f}}$, whereas the restarted policy receives a recovery instruction ℓ_r and post-restart observations but neither the pre-failure history nor the original setup demonstration.

A suitable recovery instruction must satisfy two conditions. First, it must specify an outcome consistent with the original task and failed episode, which we call semantic recoverability. Second, it must elicit appropriate subtasks and actions from the target policy, which we call policy-supported recoverability. These conditions need not coincide: as illustrated in Figure 1 (b) and (c), a semantically correct instruction may omit policy-dependent prerequisites or fall outside the behavioral support of the target policy.

Method overview. Figure 2 summarizes the complete POLICYGROUND pipeline. The framework separates recovery into Semantic Recovery Specification and Policy-Grounded Realization. Semantic Recovery Specification derives a policy-independent recovery goal $g^\star$, an optional semantic constraint ρ, and an initial recovery instruction from the recorded failed episode $\mathcal{E}_{t_\mathrm{f}}$. Policy-Grounded Realization uses Chain-of-Probe to obtain target-policy-specific evidence about each candidate before physical execution. Chain-of-Probe connects black-box high- and low-level policy responses through proxy contexts retrieved from $\mathcal{E}_{t_\mathrm{f}}$, yielding a proxy multi-step response. Throughout the method, F_SV denotes the same frozen supervisor LMM, queried with stage-specific fixed prompts. Complete prompts are provided in Appendix A.

3.2 SEMANTIC RECOVERY SPECIFICATION

Semantic Recovery Specification analyzes the original instruction ℓ and recorded failed episode $\mathcal{E}_{t_\mathrm{f}}$: $(d, g^\star, \rho, e) = F_\mathrm{SV}(\ell, \mathcal{E}_{t_\mathrm{f}}; p_\mathrm{sem})$. Here, d is the failure diagnosis, $g^\star$ is the fixed task-level recovery goal, ρ is an optional semantic constraint, and e summarizes supporting episode evidence. When a failed target selection is identified, ρ excludes that selection; otherwise, $\rho = \emptyset$.

The supervisor then generates the initial recovery instruction $\ell_\mathrm{r}^{(1)} = F_\mathrm{SV}(g^\star, \rho, d, e; p_\mathrm{init})$. It must realize $g^\star$ from the restart state while respecting ρ. Direct execution of $\ell_\mathrm{r}^{(1)}$ defines the semantic-recovery baseline, whereas POLICYGROUND refines it using responses from the frozen target policy. The fixed prompts p_sem and p_init are specified in Appendix A, and the semantic-specification procedure, including failure diagnosis and evidence aggregation, is provided in Appendix B.1.

3.3 CHAIN-OF-PROBE

Assessing whether a multi-step recovery instruction is supported by the target policy ordinarily requires closed-loop execution, because later policy inputs depend on states produced by earlier actions. Chain-of-Probe approximates this interaction without physically executing each candidate or predicting future observations with a learned dynamics model.

For the k-th candidate instruction $\ell_\mathrm{r}^{(k)}$, Chain-of-Probe alternates high-level subtask prediction and low-level action probing. For each predicted subtask $s_i^{(k)}$, it rolls out a low-level action chain by alternating action prediction and proxy-context retrieval; the final proxy context then initializes the next high-level query for $s_{i+1}^{(k)}$. The rollout terminates when the high-level policy repeats the preceding subtask or reaches the subtask horizon.

Proxy context. Let $c_{i,j}^{(k)} = (o_{i,j}^{(k)}, q_{i,j}^{(k)})$ denote the proxy context for subtask i and low-level probe step j. Each candidate starts from the failure state, $c_{1,1}^{(k)} = (o_{t_\mathrm{f}}, q_{t_\mathrm{f}})$. The high-level policy predicts the i-th subtask as

$$s_i^{(k)} \sim \pi_\mathrm{H}\left(\ell_\mathrm{r}^{(k)}, h_i^\mathrm{p}, o_{i,1}^{(k)}\right). \tag{2}$$

where the proxy history h_i^p is initialized empty and contains only subtasks predicted earlier in the same Chain-of-Probe rollout. The failed episode is used externally for proxy-context retrieval but is not provided to the target policy as pre-failure history.

Action probing. From the current proxy context, the low-level policy predicts an action chunk, whose endpoint is propagated and used to retrieve the next proxy context:

$$A_{i,j}^{(k)} \sim \pi_{\mathrm{L}}(s_i^{(k)}, c_{i,j}^{(k)}), \qquad \hat{q}_{i,j}^{(k)} = \mathrm{Propagate}(q_{i,j}^{(k)}, A_{i,j}^{(k)}), \qquad c_{i,j+1}^{(k)} = \mathrm{Retrieve}(\hat{q}_{i,j}^{(k)}, \mathcal{E}_{t_t}). \quad (3)$$

Because the future observation is unavailable without execution, $\mathrm{Retrieve}$ selects the recorded context most compatible with the predicted endpoint by matching gripper state and minimizing end-effector distance. The action chain terminates when the predicted end-effector motion and gripper state settle or when no sufficiently close proxy context can be retrieved. With $J_i^{(k)}$ valid low-level probe steps, the final proxy context $c_{i,J_i^{(k)}+1}^{(k)}$ initializes the next high-level query $c_{i+1,1}^{(k)}$.

Chain output. For a candidate with $N^{(k)}$ predicted subtasks, Chain-of-Probe returns the probed subtasks and action chains together with the retrieved proxy contexts, collectively denoted by $R_\pi^{(k)}$. This retrieval-stitched response is not an estimate of the true environment dynamics and does not guarantee physical feasibility. Instead, it provides target-policy-specific evidence about how the high-level policy interprets the candidate and whether the low-level policy produces behavior consistent with the predicted subtasks. Details of $\mathrm{Propagate}$, $\mathrm{Retrieve}$, and the completed pseudocode are provided in Appendix B.

3.4 Response Evaluation

For each candidate instruction, the frozen supervisor F_{SV} evaluates the high- and low-level policy responses produced by Chain-of-Probe using fixed evaluation prompts. At the high level, it produces a compatibility score $S_{\mathrm{H}}^{(k)} \in [0,1]$ and textual feedback $f_{\mathrm{H}}^{(k)}$. The score measures whether the predicted subtask sequence is necessary, correctly ordered, and sufficient for the semantic recovery goal. At the low level, F_{SV} independently evaluates the action response for each predicted subtask, producing $(S_{\mathrm{L},i}^{(k)}, f_{\mathrm{L},i}^{(k)})$. Each action chain is represented by an LMM-interpretable trajectory $\mathcal{T}_i^{(k)}$ containing simplified end-effector motion, gripper-state changes, and a front-view trajectory overlay (Appendix B.3).

We aggregate the low-level scores and combine the two hierarchy levels as

$$S_{\mathrm{L}}^{(k)} = \left(\prod_{i=1}^{N^{(k)}} S_{\mathrm{L},i}^{(k)} \right)^{1/N^{(k)}}, \qquad S^{(k)} = S_{\mathrm{H}}^{(k)} S_{\mathrm{L}}^{(k)}. \quad (4)$$

The geometric mean penalizes weakly supported subtasks and becomes zero if any subtask score is zero; the product similarly requires support at both hierarchy levels. If the high-level policy produces no valid subtask, we set $S_{\mathrm{H}}^{(k)} = S_{\mathrm{L}}^{(k)} = 0$. These scores are heuristic within-episode compatibility signals, not calibrated success probabilities or guarantees of physical executability. The component scores and textual feedback are used for instruction revision, while $S^{(k)}$ is used only for best-so-far candidate selection. We collect $R_\pi^{(k)}$, the hierarchy-specific scores, and their textual feedback into a structured policy-feedback record $\Phi_\pi^{(k)}$, which is passed to instruction revision.

3.5 Instruction Revision

After evaluating candidate k, we retain the best-so-far candidate $b_k = \arg\max_{r \in \{1,\dots,k\}} S^{(r)}$, breaking ties in favor of the most recent candidate. The frozen supervisor F_{SV} revises $\ell_{\mathrm{r}}^{(b_k)}$ using its policy-feedback record $\Phi_\pi^{(b_k)}$, the semantic specification $(g^\star, \rho)$, and a fixed target-policy interface description $\mathcal{D}_\pi$. The reviser also receives a record of previously attempted edits. $\mathcal{D}_\pi$ specifies task-success criteria, coordinate conventions, and supported subtask formats, but contains no episode-specific recovery information. The revision must preserve $g^\star$ and respect ρ.

Revisions primarily modify instruction structure or object reference. Structural revisions insert missing prerequisites, decompose unsupported commands, remove redundant steps, or reorder subtasks. For example, *pick up the front-left container* may become *put down the currently held container, then pick up the front-left container*. Referential revisions replace ambiguous or unsupported descriptions

with expressions compatible with the visual scene and target-policy vocabulary. Constraining revision by $\mathcal{D}_\pi$ discourages unsupported action primitives or references.

Probing, evaluation, and revision repeat until the reviser leaves the best-so-far instruction unchanged or the candidate-evaluation budget K is exhausted. The resulting instruction is returned as $\ell_{\mathrm{r}}^{\star}$. Complete pseudocode is provided in Appendix B.

4 EXPERIMENTS

We evaluate whether policy-specific evidence obtained before physical execution improves recovery over semantic correction alone, how high- and low-level policy responses contribute to this improvement, whether the resulting behavior remains aligned with the intended recovery, and what inference cost probing introduces.

4.1 EXPERIMENTAL SETUP

Protocol. We construct RoboMME-FR, a failure-recovery evaluation protocol based on RoboMME (Dai et al., 2026). RoboMME contains non-Markovian manipulation tasks in which the correct high-level decision depends on past observations, task progress, or previously observed object identities. We evaluate ten RoboMME-FR tasks: BUTTONUNMASK (BU), VIDEOUNMASK (VU), BUTTONUNMASKSWAP (BUS), VIDEOUNMASKSWAP (VUS), BINFILL (BF), PICKHIGHLIGHT (PH), VIDEOREPICK (VR), VIDEOPLACEBUTTON (VPB), VIDEOPLACEORDER (VPO), and PICKXTIMES (PX). Each task contains 50 nominal-policy episodes. Recovery is evaluated only on episodes in which the nominal policy fails. The complete selection criteria and task descriptions are provided in Appendix C.

For each task, we first execute the target policy without recovery. When it fails, we save the resulting environment state and retain the recorded failed episode $\mathcal{E}_{t_{\mathrm{f}}}$ for use by the recovery method. Every method is evaluated from the same physical post-failure state, but with a reinitialized target-policy context. The target policy receives only the generated recovery instruction and post-restart observations and policy history; the pre-failure policy history and any original setup demonstration are not re-supplied.

A recovery method may issue multiple offline black-box policy queries while evaluating candidates. After candidate selection, however, it may execute only one recovery instruction from the saved failure state. Thus, the physical environment is not reset, while the target-policy context is. We use the same context-reset interface for Chain-of-Probe and physical recovery.

Models. Our primary target system uses the released GroundSG+QwenVL variant from the MME-VLA suite accompanying RoboMME. Specifically, we use its fine-tuned Qwen3-VL-4B (Bai et al., 2025) grounded-subgoal predictor as the high-level vision-language policy and its GroundSG-conditioned $\pi_{0.5}$ checkpoint as the low-level VLA policy. Both components are kept frozen throughout our experiments. We use GPT-5.6 Luna as the frozen supervisor for semantic recovery specification, response evaluation, and instruction revision. POLICYGROUND accesses the target policies only through their standard input–output interfaces; it does not use policy weights, hidden activations, gradients, or simulator rollouts during candidate probing. We use the same prompts, candidate budgets, and convergence criteria across tasks. Complete model and implementation details are reported in Appendix D.1.

Comparison conditions. **No recovery** reports the nominal task performance of the target policy. **Semantic recovery** directly executes the initial semantic recovery instruction $\ell_{\mathrm{r}}^{(1)}$, without policy probing or revision. The complete POLICYGROUND starts from $\ell_{\mathrm{r}}^{(1)}$, iteratively evaluates and revises candidate instructions through Chain-of-Probe, and executes the returned policy-grounded instruction $\ell_{\mathrm{r}}^{\star}$. Finally, the **human-instruction reference** directly executes a human-authored, policy-compatible recovery instruction constructed from the failed history and task objective. We treat it as an oracle-style reference for recovery-instruction construction rather than as an automatic method; execution still uses the same frozen target policy.

Table 1: Recovery success (RS), end-to-end task success, and recovery-instruction generation time on ten RoboMME-FR tasks. "Failed eps." gives the number of nominal-policy failures. "Avg." is the micro-average over all failed episodes, and E2E is computed over all 500 episodes. Generation time is averaged over recovery episodes and excludes physical execution. Bold marks the better automatic recovery method in each column; ties are both bold.

	Recovery success (%)											Summary	
Method	VU	BU	VUS	BUS	PH	VR	VPB	VPO	BF	PX	Avg.	E2E	Gen. time (s)
Failed eps.	4	38	36	39	39	37	23	30	22	5	–	–	–
No recovery	–	–	–	–	–	–	–	–	–	–	–	45.4	–
Semantic recovery	0.0	10.5	5.6	10.3	2.6	**2.7**	8.7	10.0	18.2	**0.0**	7.7	49.6	**45.4**
PolicyGround	**50.0**	**47.4**	**41.7**	**12.8**	**20.5**	**2.7**	**17.4**	**16.7**	**27.3**	**0.0**	**23.4**	**58.2**	101.0
Human instruction	50.0	47.4	47.2	25.6	28.2	13.5	47.8	13.3	27.3	0.0	30.8	62.2	–

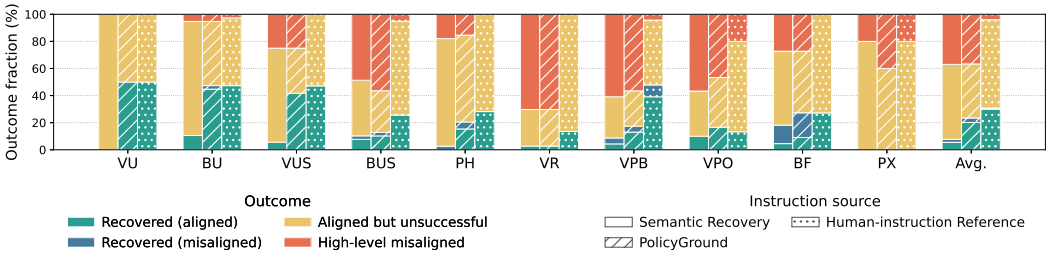


Figure 3: Recovery-outcome decomposition by task and method. HA is the sum of the aligned portions (recovered-aligned and aligned-but-unsuccessful). All nominal-policy failure episodes are included in the denominator. Recovered-but-misaligned episodes are shown separately. Human-written instructions are a reference.

Metrics. For a task with N original episodes, F nominal-policy failures, and S successful recoveries, recovery success (RS) is S/F, and end-to-end task success is $(N-F+S)/N$. High-level alignment (HA) is the fraction of the same F failed episodes whose post-resume critical-path checkpoint sequence is a valid prefix of the remaining oracle plan, independent of physical success. Episodes that emit no checkpoint, including those for which no recovery instruction is produced or recovery is skipped, are counted as misaligned. Task-specific checkpoints and grounding rules are given in Appendix D.4. Because the number of nominal-policy failures differs across tasks, "Avg." denotes the micro-average over all failed episodes. Recovery-instruction generation latency is likewise averaged over recovery episodes and excludes subsequent physical execution. Task-level latency and worst-case model-query complexity are reported in the appendix.

4.2 Recovery Performance

Table 1 compares semantic recovery and PolicyGround across the ten RoboMME-FR tasks, with human-written instructions as a human-specified reference. Because the number of initial failures varies across tasks, we report both task-level rates and a micro-average over all failed episodes.

As shown in Table 1, PolicyGround improves recovery success on eight of the ten tasks and ties semantic recovery on the remaining two. It raises micro-averaged recovery success from 7.7% to 23.4% and end-to-end task success from 49.6% to 58.2%, compared with 30.8% recovery success and 62.2% end-to-end success under human-written instructions.

Figure 3 separates physical recovery from high-level alignment. Human-written instructions achieve 95.2% HA, yet only 30.8% recovery success, revealing a substantial downstream execution bottleneck. Likewise, PolicyGround improves recovery success from 7.7% to 23.4%, while HA remains comparable to semantic recovery (60.1% vs. 60.8%). Thus, the recovery gain does not arise from higher high-level alignment; rather, it primarily reflects improved compatibility with downstream execution. The ablations below further disentangle the contributions of high- and low-level feedback.

Table 2: Ablations of policy-grounded recovery. RS measures physical recovery, whereas HA measures alignment of the executed high-level checkpoint sequence. "Avg." denotes the micro-average over all failed episodes. Human-written instructions are shown as a reference.

Variant	High-level	Low-level	Avg. RS ↑	Avg. HA ↑
Semantic recovery			7.7	60.8
High-level feedback only	✓		13.2	63.7
Low-level feedback only		✓	24.9	57.1
Full POLICYGROUND	✓	✓	23.4	60.1
Human instruction	Reference		30.8	95.2

Most remaining high-level misalignment occurs in episodes involving target swaps, particularly in VUS, BUS, and harder instances of VR, VPB, and VPO. Such cases require the supervisor to recover the changed target identity from the episode history; an incorrect semantic goal $g^\star$ cannot be corrected by subsequent policy-grounded realization. BF and PX show a similar limitation for cumulative-count reasoning.

4.3 MECHANISM ANALYSIS

Component ablations. We isolate the contributions of hierarchy-specific policy feedback and iterative refinement. *Semantic recovery* directly executes the initial instruction without probing or revision. *High-level feedback only* provides $(S_{\mathrm{H}}, f_{\mathrm{H}})$ to the reviser and uses S_{H} to maintain the best-so-far candidate. *Low-level feedback only* provides $(S_{\mathrm{L}}, \{f_{\mathrm{L},i}\})$ and uses S_{L} for best-so-far selection. The full method uses both feedback sources and maintains the best-so-far candidate using $S = S_{\mathrm{H}} S_{\mathrm{L}}$.

Table 2 reveals an RS–HA trade-off. Low-level feedback alone achieves the highest RS (24.9%) but reduces HA to 57.1%, whereas high-level feedback alone achieves the highest automatic HA (63.7%) but only 13.2% RS. The full method achieves 23.4% RS and 60.1% HA, substantially improving RS over semantic recovery while maintaining comparable HA. Thus, low-level evidence primarily drives physical recovery, while high-level evidence helps preserve oracle-consistent task progression.

A single-revision variant already captures most of the improvement, achieving 22.7% RS and 60.1% HA, compared with 23.4% RS and 60.1% HA for the full method. Among episodes producing an initial candidate, the second-revision candidate (k=3 under budget K=3) is selected in 14.2% of cases, indicating that additional refinement benefits a limited subset but exhibits rapidly diminishing returns (Appendix E.4).

4.4 INFERENCE COST

Table 1 reports wall-clock recovery-instruction generation time. Semantic recovery requires 45.4 s per recovery episode on average, whereas POLICYGROUND requires 101.0 s, approximately 2.2× longer. The additional cost arises from iterative high- and low-level policy probing, compatibility evaluation, and instruction revision. These measurements include semantic specification, candidate probing, evaluation, and revision, but exclude subsequent physical execution. Task-level latency and model-specific query counts are reported in Appendix E.2.

5 LIMITATIONS AND FUTURE WORK

POLICYGROUND cannot correct errors already introduced in the semantic recovery goal, and Chain-of-Probe only approximates closed-loop interaction: retrieved proxy contexts may differ from states produced by physical execution, so its compatibility scores do not guarantee successful execution. Our evaluation uses a single hierarchical target system and a single supervisor sample per episode; consequently, it does not quantify variation across target policies or stochastic supervisor outputs. Policy probing also increases instruction-generation latency. Future work will improve semantic reasoning and proxy-state construction, reduce probing cost, and evaluate broader target policies and real-robot settings.

AI USE STATEMENT

In this work, we used generative AI tools to assist in organizing and refining mathematical formulations developed by the authors, providing feedback on author-designed experimental methodology, supporting method implementation and code organization, and providing supplementary feedback on the authors' interpretation of experimental results. We did not use generative AI to generate synthetic datasets, develop theoretical or conceptual frameworks, provide ingredients for or assist with mathematical proofs, formulate or refine research hypotheses, translate research material, clean or reformat datasets, or perform qualitative or thematic data analysis.

Additionally, we used generative AI as an assistive tool for drafting and editing parts of the manuscript for clarity and readability, supporting software-code editing, and suggesting keywords. The technical content, experimental design, primary interpretation of results, and final wording were determined by the authors. All AI-assisted formulations, code, analyses, and manuscript edits were reviewed and verified by the authors. We take responsibility for the final content of this work, including text, claims, code, and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed information to support reproduction of our method and experiments. Section 3 defines the complete POLICYGROUND framework, and Appendix B provides pseudocode and implementation details for semantic recovery specification, Chain-of-Probe, policy-response evaluation, and iterative policy-grounded realization. Appendix A reports the complete supervisor prompts used throughout the method. Appendix C specifies the RoboMME-FR task selection and one-shot recovery protocol, while Appendix D documents the target models, hyperparameters, hardware, task-domain specifications, evaluation metrics, and high-level alignment procedure. Additional task-level results, inference-cost analysis, and revision-depth analysis are provided in Appendix E. We plan to release the implementation and evaluation code upon acceptance.

REFERENCES

Shuai Bai, Yuxuan Cai, Rui-Zhe Chen, Ke qin Chen, Xiong-Hui Chen, Zesen Cheng, Liang-Hao Deng, Wei Ding, Rongyao Fang, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Qidong Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Ming-Sheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xue-Jing Liu, Jiawei Liu, Cheng-Long Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Lingchen Meng, Xin yi Ren, Sibo Song, Yu chen Sun, Jun Tang, Jianhong Tu, Jian-Qiang Wan, Peng Wang, Pengfei Wang, Qiu-Yue Wang, Yuxuan Wang, Tianbao Xie, Yihe Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bo-Wen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Botao Zheng, Humen Zhong, Jingren Zhou, Fanxi Zhou, Jingren Zhou, Yuanzhi Zhu, and Keming Zhu. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, brian ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 27–30 Sep 2025a.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert 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, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. In *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, June 2025b.

Hongyi Chen, Yunchao Yao, Ruixuan Liu, Changliu Liu, and Jeffrey Ichnowski. Robot failure recovery using vision-language models with optimized prompts. In *2025 American Control Conference (ACC)*, pp. 1983–1988, 2025.

Nhat Chung, Taisei Hanyu, Toàn Nguyên, Huy Le, Frederick Bumgarner, Duy Minh Ho Nguyen, Khoa Vo, Kashu Yamazaki, Chase Rainwater, Tung Kieu, Anh Nguyen, and Ngan Le. Rethinking progression of memory state in robotic manipulation: An object-centric perspective. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 3407–3415, 2026.

Cristina Cornelio and Mohammed Diab. Recover: A neuro-symbolic framework for failure detection and recovery. *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 12435–12442, 2024.

Yinpei Dai, Jayjun Lee, Nima Fazeli, and Joyce Chai. Racer: Rich language-guided failure recovery policies for imitation learning. In *2025 IEEE international conference on robotics and automation (ICRA)*, pp. 15657–15664. IEEE, 2025.

Yinpei Dai, Hongze Fu, Jayjun Lee, Yue-Jiang Liu, Haoran Zhang, Jianing Yang, Chelsea Finn, Nima Fazeli, and Joyce Chai. RoboMME: Benchmarking and understanding memory for robotic generalist policies. *ArXiv*, abs/2603.04639, 2026.

David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10(2):112–122, 1973.

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

Peng Guo, Zhonghao Mai, Zheng-Tong Xu, Kai-Di Zhang, Quan Khanh Luu, Heng Zhang, Zichen Miao, Arash Ajoudani, Zachary Kingston, Qiang Qiu, and Yu She. PLanAR: Planning-language-grounded agentic reasoning for robot manipulation. *arXiv preprint arXiv:2602.01662*, 2026.

Yanjiang Guo, Yen-Jen Wang, Lihan Zha, Zheyuan Jiang, and Jianyu Chen. DoReMi: Grounding language model by detecting and recovering from plan-execution misalignment. *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 12124–12131, 2023.

Physical Intelligence, Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Gregor Balke, Kevin Black, George Bokinsky, Shi-Jie Cao, Thomas Charbonnier, Vedant Choudhary, Foster Collins, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Maitrayee Dhaka, Jared DiCarlo, Danny Driess, Michael Robert Equi, Adnan Esmail, Yun-Hao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachlan Groom, Haroun Habeeb, Hunter Hancock, Karol Hausman, Gashon Hussein, Victor Hwang, Brian Ichter, Connor Jacobsen, Szymon Jakubczak, Rowan Jen, Tim Jones, Gregg Kammerer, Ben Katz, Liyiming Ke, Mairbek Khadikov, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Brendon LeCount, Sergey Levine, Xinyu Li, Adrian Li-Bell, Vladislav Lialin, Zhong-Zhou Liang, W. K. Lim, Yao Lu, Enyu Luo, Vishnu Mano, Nandan Marwaha, A. Mongush, Liam Murphy, Suraj Nair, Tyler M. Patterson, Karl Pertsch, Allen Z. Ren, Gavin Schelske, Charvi Sharma, Baifeng Shi, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Jiaming Tang, James Tanner, Shalom Tekeste, Marcel Torne, Kyle Vedder, Quan Vuong, Anna Walling, Hao-Huan Wang, Jason Wang, Xu-Dong Wang, Christopher C. Whalen, Sir John Whitmore, Blake Williams, Charles Xu, Sukwon Yoo, Li-Li Yu, Wu Zhang, Zhuoyang Zhang, and Ury Zhilinsky. $\pi_{0.7}$: a steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 06–09 Nov 2025.

Huashuo Lei, Wenxuan Song, Huarui Zhang, Jieyuan Pei, Jiayi Chen, Haodong Yan, Han Zhao, Pengxiang Ding, Zhipeng Zhang, Lida Huang, Donglin Wang, Yan Wang, and Haoang Li. RoboMemArena: A comprehensive and challenging robotic memory benchmark. *arXiv preprint arXiv:2605.10921*, 2026.

Zijun Lin, Jiafei Duan, Haoquan Fang, Dieter Fox, Ranjay Krishna, Cheston Tan, and Bohua Wen. FailSafe: Reasoning and recovery from failures in vision-language-action models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2026.

Zeyi Liu, Arpit Bahety, and Shuran Song. REFLECT: Summarizing robot experiences for failure explanation and correction. In *7th Annual Conference on Robot Learning*, 2023.

Weijia Liufu, Xiaoyu Guo, Ruiyi Chen, Jingzhi Liu, Kaidong Zhang, Xiwen Liang, Jianqi Lin, Dawei Sun, Yuze Wang, Rongtao Xu, Bingqian Lin, Bowen Yang, Tongtong Cao, Bowen Peng, Dongyu Zhang, Guangrun Wang, Min Wang, Liang Lin, and Xiaodan Liang. RePO-VLA: Recovery-driven policy optimization for vision-language-action models. *arXiv preprint arXiv:2605.09410*, 2026.

Weifeng Lu, Minghao Ye, Zewei Ye, Tao Lin, Shuo Yang, and Bo Zhao. RoboFAC: A comprehensive framework for robotic failure analysis and correction. *ArXiv*, abs/2505.12224, 2025.

Chenyang Ma, Kai Lu, Guangyu Yang, Jiuming Liu, Shitong Xu, Bill Byrne, Ioannis Havoutis, Niki Trigoni, and Andrew Markham. Cyclevla: Proactive self-correcting vision-language-action models via subtask backtracking and minimum bayes risk decoding. *arXiv preprint arXiv:2601.02295*, 2026.

NVIDIA, Johan Bjorck, Nikita Cherniadev Fernando Castañeda, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1(3):244–256, 1972.

Lucy Xiaoyang Shi, brian ichter, Michael Robert Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. Hi Robot: Open-ended instruction following with hierarchical vision-language-action models. In *Forty-second International Conference on Machine Learning*, 2025.

Suyeon Shin, Juwon Kim, Hyeonbin Park, Hyunseo Kim, Hyundo Lee, Hyung-Sin Kim, and Byoung-Tak Zhang. Back to the familiar future: Failure recovery for vla policies via pre-imagined milestone selection. *arXiv preprint arXiv:2606.09258*, 2026.

Marta Skreta, Zihan Zhou, Jia Yuan, Kourosh Darvish, Alán Aspuru-Guzik, and Animesh Garg. RePLan: Robotic replanning with perception and language models. *ArXiv*, abs/2401.04157, 2024.

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. MemER: Scaling up memory for robot control via experience retrieval. In *The Fourteenth International Conference on Learning Representations*, 2026.

Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montse Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauzá, Michiel Blokzijl, Steven Bohez, Konstantinos Bousmalis, Anthony Brohan, Thomas Buschmann, Arunkumar Byravan, Serkan Cabi, Ken Caluwaerts, Federico Casarini, Oscar Chang, José Enrique Chen, Xi Chen, Hao-Tien Lewis Chiang, Krzysztof Choromanski, David B. D'Ambrosio, Sudeep Dasari, Todor Davchev, Coline Devin, Norman Di Palo, Tianli Ding, Adil Dostmohamed, Danny Driess, Yilun Du, Debidatta Dwibedi, Michael Elabd, Claudio Fantacci, Cody Fong, Erik Frey,

Chuyuan Kelly Fu, Marissa Giustina, Keerthana Gopalakrishnan, Laura Graesser, Leonard Hasenclever, Nico-351 las Heess, Brandon Hernaez, Alex Herzog, R. Alex Hofer, Jan Humplik, Atil Iscen, Mithun George Jacob, Deepali Jain, Ryan C. Julian, Dmitry Kalashnikov, Mustafa Emre Karagozler, Stefani Karp, Chase Kew, Jerad Kirkland, Sean Kirmani, Yuheng Kuang, Thomas Lampe, Antoine Laurens, Isabel Leal, Alex X. Lee, Tsang-Wei Edward Lee, Jacky Liang, Yi-Xin Lin, Sharath Maddineni, Anirudha Majumdar, Assaf Hurwitz Michaely, Robert Moreno, Michael Neunert, Francesco Nori, Carolina Parada, Emilio Parisotto, Peter Pastor, Acorn Pooley, Kanishka Rao, Krista Reymann, Dorsa Sadigh, Stefano Saliceti, Pannag R. Sanketi, Pierre Sermanet, Dhruv Shah, Mohit Sharma, Kathryn Shea, Charles Shu, Vikas Sindhwani, Sumeet Singh, Radu Soricut, Jost Tobias Springenberg, Rachel Sterneck, Razvan Surdulescu, Jie Tan, Jonathan Tompson, Vincent Vanhoucke, Jake Varley, Grace Vesom, Giulia Vezzani, Oriol Vinyals, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Fei Xia, Ted Xiao, Annie Xie, Jinyu Xie, Peng Xu, Sichun Xu, Ying Xu, Zhuo Xu, Yuxiang Yang, Rui Yao, Sergey Yaroshenko, Wen-Hao Yu, Wentao Yuan, Jingwei Zhang, Tingnan Zhang, Allan Zhou, and Yuxiang Zhou. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

Marcel Torne, Karl Pertsch, Homer Rich Walke, Kyle Vedder, Suraj Nair, Brian Ichter, Allen Z. Ren, Hao-Huan Wang, Jiaming Tang, Kyle Stachowicz, Karan Dhabalia, Michael Equi, Quan Ho Vuong, Jost Tobias Springenberg, Sergey Levine, Chelsea Finn, and Danny Driess. MEM: Multi-scale embodied memory for vision language action models. *ArXiv*, abs/2603.03596, 2026.

Zezhong Wang, Xingshan Zeng, Weiwen Liu, Yufei Wang, Liangyou Li, Yasheng Wang, Lifeng Shang, Xin Jiang, Qun Liu, and Kam-Fai Wong. Chain-of-Probe: Examining the necessity and accuracy of CoT step-by-step. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2586–2606, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.140.

Zixuan Wang, Yuxin Chen, Yuqi Liu, Jinhui Ye, Pengguang Chen, Changsheng Lu, Shu Liu, and Jiaya Jia. VP-VLA: Visual prompting as an interface for vision-language-action models. *arXiv preprint arXiv:2603.22003*, 2026.

Xianchao Zeng, Xinyu Zhou, Youcheng Li, Jiayou Shi, Tianle Li, Liangming Chen, Lei Ren, and Yong-Lu Li. Diagnose, correct, and learn from manipulation failures via visual symbols. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 42386–42395, 2026.

Fan Zhang, Seongbin Park, Baharan Mirzasoleiman, Shariar Talebi, and Nader Sehatbakhsh. ProbeAct: Probe-guided training-free failure recovery in vision-language-action models. *arXiv preprint arXiv:2606.09740*, 2026.

Peiyuan Zhi, Zhiyuan Zhang, Yu Zhao, Muzhi Han, Zeyu Zhang, Zhitian Li, Ziyuan Jiao, Baoxiong Jia, and Siyuan Huang. Closed-loop open-vocabulary mobile manipulation with GPT-4V. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4761–4767. IEEE, 2025.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 06–09 Nov 2023.

A SUPERVISOR PROMPTS

This appendix reports the prompts used by the frozen supervisor LMM F_{SV}. Text in angle brackets denotes a runtime value. Each call consists of the system prompt below, its listed user-side inputs, and the attached images or video where specified. We denote the prompts as p_{sub} (per-subtask outcome diagnosis), p_{ep} (whole-episode diagnosis), p_{agg} (recovery-goal aggregation), p_{init} (initial recovery-task generation), $p^{\mathrm{H}}_{\mathrm{eval}}$ / $p^{\mathrm{L}}_{\mathrm{eval}}$ (high- and low-level compatibility evaluation), and p_{rev} (instruction revision).

A.1 SEMANTIC RECOVERY SPECIFICATION

The semantic analysis pipeline p_{sem} comprises three supervisor prompts: p_{sub} for per-subtask outcome diagnosis, p_{ep} for whole-episode diagnosis, and p_{agg} for recovery-goal aggregation. The supervisor then generates the initial recovery instruction $\ell_{\mathrm{r}}^{(1)}$ using p_{init}.

Per-subtask outcome diagnosis (p_{sub}). User-side inputs: general task context; task goal ℓ; issued-subtask rollout; subtask under review; interval bounds; optional setup demonstration frames; and uniformly sampled, synchronized front and wrist frames from the interval.

Per-subtask Outcome Diagnosis

```
Task:
Judge whether ONE issued subtask was completed over its interval, and describe its
     outcome and final state.

Inputs:
- Task goal l, general task context, issued-subtask rollout, subtask under review with
     interval bounds, optional setup demonstration frames, and interval frames.
- Frames within each camera stream are chronological; matching `video_frame_idx` values
      are synchronized, and FINAL is the last frame. Frame-label `gripper=open|closed`
     is proprioception ground truth.

Rules:
- Score from the interval frames only; setup demonstration frames, rollout, and episode
     -FAILURE text are context. Read the starting state from the first interval frame(s
     ).
- Expand the subtask into frame-checkable `success_requirements` covering every action
     clause and target-identifying phrase, without adding content. Express identity
     requirements as observable conditions, not restatements of the claim. They require
      positive visual evidence in any interval frame; concealment or lack of
     contradiction is insufficient.
- Mark `done` only when every requirement is met. A mismatched target or observed
     attribute conflict is `failed`, even when the motor action succeeds.
- Use `uncertain` when the interval frames do not establish whether every requirement
     was met. Do not infer an action, contact, state change, or target identity from
     the instruction, a closed gripper, or ambiguous/occluded imagery; state only what
     is visually supported.
- Treat `<y, x>` in the subtask text as a spatial hint only, not identity or success
     evidence. Reuse the subtask's own ground when naming its target; otherwise do not
     invent coordinates.
- Combine synchronized camera views: positive evidence in either view counts, while
     occlusion or absence in one view does not override visibility in another.
- In `final_state`, separate holding/contact from pickup/lift: a closed gripper alone
     is insufficient, but a visibly contacted or enclosed object may be reported as
     held; report pickup only when the identified container is observed leaving the
     table. Determine object identity and attributes from the full interval, and do not
      infer them from the instruction.
- `outcome` states the observed action and effect. `final_state` is a present-tense
     FINAL snapshot of gripper state, held/contacted object state, rough arm pose, and
     visible state of the manipulated object. Use no time deixis or unverified claims.

Status:
- done -- every success_requirement is met by end of interval.
- failed -- at least one is not (wrong target, unmet identifying phrase, incomplete
     action, or the scene was not ready).
- uncertain -- the interval evidence does not establish whether every requirement was
     met.

Output: Return only this JSON in one ```json code block:
{
      subtask_name: <the name of the subtask>,
      success_requirements: [
            <observable necessary condition from the subtask text>,
            <identifying-phrase condition needing positive scene evidence in some frame
                 >,
```

```
            <...>
      ],
      reasoning: <brief evidence for each requirement: met or not>,
      subtask_status: <done | failed | uncertain>,
      outcome: <one sentence: what actually happened (action + effect)>,
      final_state: <present-tense FINAL-frame snapshot>
}
```

Whole-episode diagnosis (p_{ep}). User-side inputs: general task context; task goal ℓ; issued-subtask rollout; and the full observation video. A setup demonstration, when available, precedes the observation video.

Whole-episode Diagnosis

```
Task:
The episode is known to have failed. Identify visually verified evidence for a standing
     unmet task requirement, and describe the final state.

Inputs:
- Task goal l, general task context, issued-subtask rollout, and the full episode video
    .
- The video may begin with a setup demonstration clip. It is an example shown to the
    policy, not robot execution. Observation frames are chronological front-camera
    images. Their labels provide `env_step`, `t_sec`, and `video_frame_idx`; `gripper=
    open|closed`, when present, is recorded proprioception ground truth. FINAL is the
    last observation frame.

Rules:
- Score the complete observation video, not the setup demonstration or the episode-
    FAILURE text. Read the starting state from the first observation frame(s) and the
    restart state from FINAL.
- Expand l into frame-checkable `success_requirements` covering every action clause and
     target-identifying phrase, without adding content. Express identity requirements
    as observable conditions, not restatements of the claim. They require positive
    visual evidence in the observation video; concealment or lack of contradiction is
    insufficient.
- In `failure_diagnosis`, cite only a task requirement that the video clearly shows is
    unmet. Do not guess that an action, contact, state change, or target identity
    occurred or did not occur from the instruction, a closed gripper, or ambiguous/
    occluded imagery. When the video does not reveal a specific unmet requirement, say
     that the failure reason is not visually verified.
- Treat `<y, x>` in issued subtasks as spatial hints only, not identity or success
    evidence. Do not invent coordinates.
- In `final_state`, separate holding/contact from pickup/lift: a closed gripper alone
    is insufficient, but a visibly contacted or enclosed object may be reported as
    held; report pickup only when the identified container is observed leaving the
    table. Determine object identity and attributes from the video, and do not infer
    them from the instruction.
- `failure_diagnosis` states the present mismatch with l, without a recovery plan. `
    final_state` is a present-tense FINAL snapshot. Use no time deixis or unverified
    claims.

Output: Return only this JSON in one ```json code block:
{
      success_requirements: [
            <observable necessary condition from l>,
            <identifying-phrase condition needing positive scene evidence>,
            <...>
      ],
      reasoning: <brief visual evidence for the verified unmet requirement, or that
          none is visually verified>,
      failure_diagnosis: <one sentence: visually verified standing mismatch with l, or
          that the failure reason is not visually verified>,
      final_state: <present-tense FINAL-frame snapshot>
}
```

Recovery-goal aggregation (p_{agg}). User-side inputs: general task context; task goal ℓ; per-subtask outcomes with grounding coordinates removed; the auxiliary whole-episode diagnosis; and an optional failed grounded-subtask reference. No video is attached. The outputs include the failure diagnosis d, recovery goal $g^{\star}$, and semantic constraint ρ.

Recovery-goal Aggregation

```
The episode is already a FAILURE. From the per-subtask outcomes, an auxiliary whole-
    episode diagnosis, the task goal, and the general task context only (no video),
    state one failure diagnosis and one recovery goal g*.

The main subtask sequence has grounding coordinates stripped on purpose because they
    drift. A failed grounded subtask may be included separately as non-strict
    reference evidence: use its semantic target and observed outcome, but do not
    evaluate coordinate precision or preserve its coordinates as exact geometry.

Rules:
- Treat the general task context and the per-subtask outcomes as ground truth; do not
    re-litigate them. The whole-episode diagnosis is independent auxiliary visual
    evidence: use it to resolve cross-interval progress and FINAL state, but do not re
    -score individual subtasks or override a per-subtask outcome without direct FINAL-
    state evidence.
- Surface level only. One diagnosis and one g*, not alternatives.
- `failure_diagnosis`: the standing mismatch with l. Present tense. No plan.
- `recovery_goal` is g*: the remaining success conditions of l as judged from the
    episode-end / failure-moment restart. Drop only clauses whose effects are already
    achieved per the per-subtask outcomes; keep every still-outstanding success
    condition (if l has several clauses, list the remaining ones in l's order). Do not
     paraphrase the diagnosis, do not write a recovery plan or concrete instruction,
    do not add grounding coordinates, and do not invent requirements beyond l.
- `recovery_constraints`: one concise semantic negative constraint that prevents
    repeating the diagnosed failed target selection, when the failed grounded
    reference supports one; otherwise an empty string. Scope it to the affected goal
    condition. Identify the failed object by semantic evidence such as its observed
    contents, appearance, or relative location; any supplied ground is reference
    evidence only, not an exact coordinate to reproduce or avoid numerically. This is
    not an additional success condition or action.
- No extra fields.

Output only this JSON in one ```json code block:
{
      reasoning: <brief evidence: which per-subtask outcome triggered the failure;
          which l clauses remain>,
      failure_diagnosis: <one sentence>,
      recovery_goal: <remaining success conditions from restart; one clause, or several
           joined in l's order>,
      recovery_constraints: <semantic negative target-selection constraint, or empty
          string>
}
```

Initial recovery-task generation (p_{init}). User-side inputs: general task context; original task ℓ; optional task and subtask templates; $g^\star$; semantic constraint ρ; a restart-state snapshot; and the complete episode video.

Initial Recovery-task Generation

```
The episode failed. The robot restarts from the episode-end / failure-moment scene (
    where the recording stopped) -- the same state the probe will start from. Do not
    assume a rewind to an earlier subtask.

From that restart state, write ONE recovery task `\ell_\mathrm{r}^{(l)}`: a concrete
    imperative that realizes g*. Do not emit alternatives.

Inputs: task context, original l, optional in-domain task and subtask templates (
    wording guidance), recovery goal g* (remaining success conditions from restart;
    fixed for later revise), the restart-state snapshot, and the full observation
    video. When a setup demonstration clip is attached, it comes first (example video
    shown to the policy at start and kept visible during execution; not robot
    execution), then the observation video.

Rules:
- Treat task context and g* as ground truth. Instantiate every condition in g*, in
    order, without adding conditions outside it; g* already excludes achieved
    conditions.
- Treat recovery constraints as semantic negative evidence for target selection, not as
     extra actions or success conditions. Do not repeat the failed object selection.
    Grounds quoted in a constraint are approximate historical references only:
    identify objects afresh from the restart state and video rather than comparing
    coordinates literally.
- Use the restart-state snapshot and video to determine the current layout. For an
    ambiguous target, estimate `<y, x>` fresh in the 0..1000 range (y top-to-bottom, x
    left-to-right), not raw image pixels; convert pixels as `round(1000 * pixel /
    image_size)`. Write every ground as `<y, x>` and copy the selected pair unchanged
    from `reasoning` into `recovery_task`.
```

```
- Treat task and subtask templates as optional wording guidance. Prefer a close
    template when it fits, but adapt or omit it when correctness, completeness, or
    executability requires it. If used, place grounding in its indicated slot.
- Keep the instruction concise, executable, and safe, using the syntax and vocabulary
    of l when suitable. Do not write a correction of l or a plan beyond g*.
- In `reasoning`, briefly justify coverage of g* and target selection. Do not provide
    alternatives.

Output only this JSON in one ```json code block:
{
      restart_state: <one to two sentences: what is true at the failure-moment restart
          >,
      reasoning: <brief evidence: how this instruction covers all of g* and why these
          objects are selected>,
      recovery_task: <the recovery task instruction \ell_\mathrm{r}^{(1)} realizing g*>
}
```

A.2 CANDIDATE EVALUATION

The evaluator prompts return integer scores in $[0, 10]$; we divide these scores by 10 before computing the normalized compatibility scores S_{H} and $S_{\mathrm{L},i}$ used in the main text.

High-level compatibility evaluation ($p^{\mathrm{H}}_{\mathrm{eval}}$). **User-side inputs:** general task context; $g^\star$; recovery constraints; restart state; candidate recovery instruction; emitted subtasks; and optional synchronized front/wrist start-image pairs.

High-level Compatibility Evaluation

```
Task:
Score how well the emitted high-level subtask sequence realizes the recovery task.

Inputs:
- General task context, recovery goal g*, recovery constraints, restart state, recovery
     task instruction, and emitted subtasks in order.
- Optional synchronized front/wrist start-image pair for each subtask.

Rules:
- Judge semantic necessity, sufficiency, workable order, and named-object consistency.
    Every emitted unit must be needed and no required unit may be missing.
- Use images and the current state only to assess start conditions and workable order.
    Do not assess whether a ground identifies the correct physical instance.
- Use recovery constraints only when a subtask text explicitly names the excluded
    target. An excluded object merely appearing in a start image is not a mismatch.
- Do not judge low-level motion or propose a better sequence.

Scoring (integer 0..10):
- 10: necessary and sufficient, correct order, and named targets match. Minor wording
    differences are fine.
- 7-9: right plan with a small omission, extra hedge, or slightly vague target
    description.
- 4-6: partial: a needed step is missing, order is mixed, or an irrelevant step is
    included, but the sequence still overlaps the task. Object identity is roughly
    right.
- 1-3: mostly wrong order, explicit target mismatch, or a fragment of the task.
- 0: empty, unrelated, or the reverse of what the task asks.

Output: Return only this JSON in one ```json block:
{
      reasoning: <brief evidence: necessity, sufficiency, order, and named-target
          consistency>,
      score: <integer 0..10>
}
```

Low-level compatibility evaluation ($p^{\mathrm{L}}_{\mathrm{eval}}$). **User-side inputs:** general task context; restart state; emitted subtasks; their synchronized start-image pairs; piecewise-linear workspace-frame displacement vectors; gripper-event intervals; and coverage status. The default trajectory input is vectors; three-view trajectory plots are an ablation.

Low-level Compatibility Evaluation

```
Task:
Score how well the observed low-level motion realizes each emitted subtask.

Inputs:
- Subtask text and optional synchronized front/wrist start-image pair.
- Commanded end-effector path in workspace-frame millimetres and commanded gripper
    states over the same time.
- Coverage status: settled or truncated. Each path line is one commanded displacement
    in millimetres. t=0 is the start image, t=1 is the end of what was observed. '(
    hold)' means a few millimetres or less.

Rules:
- Judge only this subtask's action and image-visible target, not whether it is the
    right high-level step. Do not infer target identity from coordinates.
- Compare both path-to-target motion and the required gripper change: open (+1+1),
    close (-1-1), release (-1+1), or grasp (+1-1).
- Infer the target-relative effect required by the subtask from the start image and
    workspace directions. When shown, compare the grasp/release marker location with
    the marked or image-visible target. Use a marker only as spatial target evidence,
    not proof of semantic identity.
- For subtask 1, use the restart-state snapshot unless its start image contradicts it;
    otherwise use each start image. An apparent completed action counts only with
    direct evidence it applies to the target. A coordinate or gripper proximity is not
     identity evidence.
- If coverage is truncated, score only the observed prefix and do not require unseen
    later phases.
- Do not propose a better motion.

Scoring (integer 0..10):
- 10: path and gripper realize the required target-relative effect.
- 7-9: the action is directed toward the target but is loose or a correctly directed
    truncated prefix.
- 4-6: partial target-relative action; needed translation or gripper change is
    incomplete or mixed with unrelated motion.
- 1-3: mostly unrelated or opposite motion/gripper, including an action on a
    distinguishable non-target object.
- 0: no required action, motion away from the target, or visual evidence contradicts
    the claimed target action.

Output: Return only this JSON in one ```json block:
{
      items: [{
            index: <1-based subtask index>,
            subtask: <copy the subtask text>,
            observed: <one line: start image, path, and gripper>,
            score: <integer 0..10>,
            reasoning: <brief evidence: path vs target, gripper vs action; note
                 truncation if applicable>
      }]
}
```

A.3 Instruction Revision

Policy-grounded instruction revision (p_{rev}). User-side inputs: general task context; original task ℓ; optional task and subtask templates; invariant $g^{\star}$; semantic constraint ρ; restart state; the best-scoring candidate; S / S_{H} / S_{L} evidence (an ablation may withhold one channel); emitted subtask sequence when provided; previous edits; and the complete episode video.

Policy-grounded Instruction Revision

```
Task:
Revise the base recovery task for a frozen VLA using only changes supported by the
    probe evidence. This is not a free rewrite.

Inputs:
- General task context, original task l, optional task/subtask templates (wording
    guidance), invariant recovery goal g*, semantic recovery constraints, and restart-
    state snapshot.
- Base recovery task, available S_\pi / S_high / S_low evidence (an ablation may
    withhold one channel), emitted subtask sequence when provided, and previous edits.
- Episode video for current layout and corroboration. A setup demonstration, when
    present, appears first and is not robot execution.

Operators:
```

```
- STRUCTURAL: change the sequence of task units by inserting, dropping, reordering,
    splitting, or merging. Do not name or rebind entities.
- REFERENTIAL: rebind the entity, state, or ground named by a unit, including naming an
     inserted unit; do not change the sequence. Preserve useful identity cues. For an
    ambiguous target, estimate `<y, x>` fresh in the 0..1000 range (y top-to-bottom, x
    left-to-right), using the template's grounding slot when applicable. A wrong
    coordinate space is a referential defect, not a small adjustment; convert it to
    0..1000. Write every ground as `<y, x>` and copy the selected pair unchanged
    between the reason and `revised_task`.

Domain alignment:
After operator edits, project wording without changing action meanings, sequence, or
    bindings; this projection cannot remedy an S_low failure. Prefer task/subtask
    templates as the unit inventory, granularity, and wording. Depart only when
    necessary for correctness, completeness, or executability, and do not add finer-
    grained units when a template suffices.

Rules:
- Treat general task context and g* as ground truth. Preserve every condition in g*, in
     order; g* already excludes achieved conditions. Instantiate g* rather than
    copying it verbatim.
- Treat recovery constraints as semantic negative evidence for target selection, not as
     extra actions or success conditions. Do not repeat the failed object selection.
    Any quoted ground is an approximate historical reference only; identify objects
    afresh from the restart state and video instead of comparing coordinates literally
    .
- Address only the evidenced causes that are provided. When both S_high and S_low are
    present, revise in causal order: earliest inadequately executed unit, STRUCTURAL
    before REFERENTIAL. If its starting state is unmet, first consider inserting an
    available template unit that establishes it. Do not revise later units until the
    preceding prefix is executable enough to reach them. Do not invent causes for a
    withheld channel.
- Treat small coordinate-only adjustments as low priority unless the current ground
    selects the wrong target; a high S_high only shows agreement with the possibly
    incomplete base task.
- Prefer the smallest effective revision. If no new change is directly supported by the
     evidence, keep the current base task: copy it unchanged into `revised_task` and
    set all change fields to `none`. Do not invent a structural or referential edit
    just to make `revised_task` differ from the base task. Any applied change must not
    repeat a failed previous edit.
- Keep the revised task executable and safe from the restart state.

Output: Return only this JSON in one ```json block:
{
      revised_task: <the revised recovery task, or the unchanged base task when no
          evidence-supported revision exists>,
      structural: {
            change: <sequence edit only; 'none' if not applied>,
            reason: <why this structural edit was or was not needed>
      },
      referential: {
            change: <entity/state rebind (incl. new units); 'none' if not applied>,
            reason: <why this referential edit was or was not needed>
      },
      domain_alignment: {
            change: <in-domain wording change; 'none' if not applied>,
            reason: <why this projection was or was not needed>
      }
}

Fill every field. Use `none` for an unused change and explain why in its reason.
```

B Detailed Algorithms

B.1 Semantic Recovery Specification

The pipeline p_{sem} expands the black-box call $(d, g^\star, \rho, e) = F_{\text{SV}}(\ell, \mathcal{E}_{t_f}; p_{\text{sem}})$ into three supervisor calls using p_{sub}, p_{ep}, and p_{agg}, whose complete prompts are provided in Appendix A. Issued high-level turns are first grouped into intervals by merging consecutive subtasks that match after stripping grounding coordinates. Per-interval outcome diagnosis and an auxiliary whole-episode diagnosis run in parallel; aggregation then emits a single failure diagnosis d, recovery goal $g^\star$, and optional constraint ρ. The evidence record e packages the per-interval outcomes, the whole-episode witness, and the restart-state snapshot used by later stages. Algorithm 1 summarizes the procedure; initial realization with p_{init} follows as in Section 3.2.

Algorithm 1 Semantic Recovery Specification

Require: Original task ℓ, failed episode $\mathcal{E}_{t_\mathrm{f}}$

1: Group issued subtasks of $\mathcal{E}_{t_\mathrm{f}}$ into intervals $\{I_m\}_{m=1}^{M}$ (merge consecutive turns equal after stripping grounds)
2: **for** $m = 1, \ldots, M$ **do**
3: $\quad u_m \leftarrow F_{\mathrm{SV}}(\ell, I_m, \text{interval frames}; p_{\mathrm{sub}})$
4: **end for**
5: $e_{\mathrm{ep}} \leftarrow F_{\mathrm{SV}}(\ell, \text{full observation video}; p_{\mathrm{ep}})$
6: $(d, g^\star, \rho) \leftarrow F_{\mathrm{SV}}(\ell, \{u_m\}_{m=1}^{M}, e_{\mathrm{ep}}; p_{\mathrm{agg}})$ ▷ aggregate; no video
7: $e \leftarrow$ evidence from $\{u_m\}$, e_{ep}, and the restart-state snapshot
8: **return** $(d, g^\star, \rho, e)$

B.2 Chain-of-Probe

State propagation and context retrieval. Given a current robot state q and an action chunk A, $\mathrm{Propagate}(q, A)$ returns the predicted endpoint robot state. Under the absolute-action low-level policy used in our experiments, this is the final commanded arm configuration in the chunk (with the corresponding end-effector pose used for retrieval). Given a predicted endpoint $\hat{q}$ and failed episode $\mathcal{E}$, $\mathrm{Retrieve}(\hat{q}, \mathcal{E})$ selects a recorded proxy context by matching binary gripper state (open/closed) and minimizing end-effector distance in millimetres, excluding the current frame ν_{cur}. Let $\mathcal{I}(\mathcal{E})$ denote the set of recorded time indices in $\mathcal{E}$, and let

$$\mathcal{I}(\hat{q}) = \{\nu \in \mathcal{I}(\mathcal{E}) : b(q_\nu) = b(\hat{q}),\ \nu \neq \nu_{\mathrm{cur}}\}, \tag{5}$$

then

$$\nu^\star = \arg\min_{\nu \in \mathcal{I}(\hat{q})} \left\| x_{\mathrm{EE}}(\hat{q}) - x_{\mathrm{EE}}(q_\nu) \right\|_2, \tag{6}$$

where $b(q)$ denotes the binary gripper state and $x_{\mathrm{EE}}(q)$ the end-effector position associated with robot state q. Retrieval fails if $\mathcal{I}(\hat{q})$ is empty or the residual exceeds δ_{ret}; otherwise $\mathrm{Retrieve}(\hat{q}, \mathcal{E}) = (o_{\nu^\star}, q_{\nu^\star})$.

An action chunk is *settled* when the start-centered enclosing radius of its commanded end-effector path is at most δ_{settle} and the commanded gripper open/closed state is unchanged over the chunk. When the action chain stops without a usable next retrieval (settle, retrieval failure, or hop budget), the outer subtask chain continues from the last probed context. Grounding coordinates emitted by π_{H} are rescaled to the image-coordinate convention expected by π_{L} before action probing. In our implementation, high-level grounds use the $[0, 1000]$ coordinate range, following the released policy interface, and are linearly rescaled to the $[0, 255]$ range expected by the low-level policy. When advancing contexts, observations come from the retrieved frame, while the low-level proprioceptive state may be set to the commanded endpoint from the preceding hop (chain-consistent state).

Algorithm 2 summarizes the procedure. Indices match the main text (i for subtasks, j for low-level hops). A repeated subtask is detected after stripping grounding coordinates and does not enter R_π.

B.3 Low-level Trajectory Representation

Raw action chunks are difficult for the supervisor LMM to interpret directly. We therefore convert each probed action chain into a compact representation of its predicted physical behavior.

We first propagate the action chunks to obtain the corresponding end-effector (EE) trajectory in the robot workspace together with the gripper-state sequence. The EE path is then simplified using the Ramer–Douglas–Peucker (RDP) algorithm (Ramer, 1972; Douglas & Peucker, 1973), which approximates the trajectory with a small number of key waypoints while preserving major changes in motion direction. The simplified trajectory is represented as a sequence of workspace-frame displacement vectors, together with gripper-state changes.

To provide spatial context for the motion, we additionally project the simplified EE trajectory onto the front-view observation using the known camera calibration. The projected trajectory is rendered on the image and provided to the supervisor together with the displacement vectors and gripper-state sequence. The numerical representation summarizes the commanded motion, while the image

Algorithm 2 Chain-of-Probe

Require: Recovery instruction ℓ_{r}, failed episode $\mathcal{E}_{t_{\mathrm{f}}}$, target policies $(\pi_{\mathrm{H}}, \pi_{\mathrm{L}})$, subtask horizon N, action-chain horizon M, settle radius δ_{settle}, retrieval threshold δ_{ret}

$c_{1,1} \leftarrow (o_{t_{\mathrm{f}}}, q_{t_{\mathrm{f}}})$ $\quad h_1^{\mathrm{P}} \leftarrow \emptyset$; $\quad R_\pi \leftarrow \emptyset$
for $i = 1, \ldots, N$ **do**
  $s_i \leftarrow \pi_{\mathrm{H}}(\ell_{\mathrm{r}}, h_i^{\mathrm{P}}, o_{i,1})$
  if $i > 1$ **and** s_i repeats s_{i-1} (ignoring grounds) **then**
    break ▷ not appended to R_π
  end if
  if s_i is unusable **then**
    Append $(s_i, \{c_{i,1}\}, \emptyset)$ to R_π
    break
  end if
  $s_i' \leftarrow$ coordinate-converted version of s_i
  $J_i \leftarrow 0$; *stop* $\leftarrow$ false
  for $j = 1, \ldots, M$ **do**
    $A_{i,j} \leftarrow \pi_{\mathrm{L}}(s_i', c_{i,j})$
    $\hat{q}_{i,j} \leftarrow \mathrm{Propagate}(q_{i,j}, A_{i,j})$
    $J_i \leftarrow j$
    if $A_{i,j}$ is settled (radius $\leq \delta_{\mathrm{settle}}$, gripper unchanged) **then**
      $c_{i,j+1} \leftarrow c_{i,j}$; *stop* $\leftarrow$ true
      break
    end if
    $\mathcal{I}_{i,j} \leftarrow \{\nu \in \mathcal{I}(\mathcal{E}_{t_{\mathrm{f}}}) \mid b(q_\nu) = b(\hat{q}_{i,j}),\ \nu \neq \nu_{i,j}\}$
    if $\mathcal{I}_{i,j} = \emptyset$ **then**
      $c_{i,j+1} \leftarrow c_{i,j}$; *stop* $\leftarrow$ true
      break
    end if
    $\nu^\star \leftarrow \arg\min_{\nu \in \mathcal{I}_{i,j}} \|x_{\mathrm{EE}}(\hat{q}_{i,j}) - x_{\mathrm{EE}}(q_\nu)\|_2$
    if $\|x_{\mathrm{EE}}(\hat{q}_{i,j}) - x_{\mathrm{EE}}(q_{\nu^\star})\|_2 > \delta_{\mathrm{ret}}$ **then**
      $c_{i,j+1} \leftarrow c_{i,j}$; *stop* $\leftarrow$ true
      break
    end if
    $c_{i,j+1} \leftarrow (o_{\nu^\star}, q_{\nu^\star})$
  end for
  if not *stop* **then**
    $c_{i,J_i+1} \leftarrow$ furthest context visited from $c_{i,1}$ along the hop path
  end if
  Append $(s_i, \{c_{i,j}\}_{j=1}^{J_i+1}, \{A_{i,j}\}_{j=1}^{J_i})$ to R_π
  $c_{i+1,1} \leftarrow c_{i,J_i+1}$; $h_{i+1}^{\mathrm{P}} \leftarrow h_i^{\mathrm{P}} \cup \{s_i\}$
end for
return R_π

projection helps the supervisor assess whether the motion is directed toward the visually specified target.

The low-level evaluator receives the subtask text, retrieved proxy contexts, trajectory representation, and projected front-view image, and returns a compatibility score and textual feedback. The projection is used only as visual context for semantic evaluation; we do not compute pixel-level geometric scores from the projected trajectory. The RDP tolerance and trajectory-conversion procedure are fixed across all experiments.

B.4 Policy-Grounded Realization

Algorithm 3 summarizes the complete iterative recovery procedure. $\mathrm{Evaluate}_{\mathrm{SV}}$ denotes the high- and low-level supervisor evaluations defined in Section 3.4; it is not a separately trained evaluator.

Algorithm 3 POLICYGROUND

Require: Original task ℓ, failed episode $\mathcal{E}_{t_\mathrm{f}}$, target policy $\pi = (\pi_\mathrm{H}, \pi_\mathrm{L})$, target-policy interface description $\mathcal{D}_\pi$, candidate budget K

$(d, g^\star, \rho, e) \leftarrow F_\mathrm{SV}(\ell, \mathcal{E}_{t_\mathrm{f}}; p_\mathrm{sem})$
$\ell_\mathrm{r}^{(1)} \leftarrow F_\mathrm{SV}(g^\star, \rho, d, e; p_\mathrm{init})$
$S_\mathrm{best} \leftarrow -\infty$
$\mathcal{P} \leftarrow \emptyset$
for $k = 1, \ldots, K$ **do**
  $R_\pi^{(k)} \leftarrow \mathrm{CoP}(\ell_\mathrm{r}^{(k)}, \pi, \mathcal{E}_{t_\mathrm{f}})$
  $(S_\mathrm{H}^{(k)}, S_\mathrm{L}^{(k)}, f_\mathrm{H}^{(k)}, f_\mathrm{L}^{(k)}) \leftarrow \mathrm{Evaluate}_\mathrm{SV}(g^\star, \rho, \ell_\mathrm{r}^{(k)}, R_\pi^{(k)})$
  $S^{(k)} \leftarrow S_\mathrm{H}^{(k)} S_\mathrm{L}^{(k)}$
  if $S^{(k)} \geq S_\mathrm{best}$ **then**
    $b \leftarrow k$
    $S_\mathrm{best} \leftarrow S^{(k)}$
  end if
  if $k = K$ **then**
    return $\ell_\mathrm{r}^{(b)}$
  end if
  $\widetilde{\ell} \leftarrow F_\mathrm{SV}(g^\star, \rho, \ell_\mathrm{r}^{(b)}, R_\pi^{(b)}, S_\mathrm{H}^{(b)}, S_\mathrm{L}^{(b)}, f_\mathrm{H}^{(b)}, f_\mathrm{L}^{(b)}, \mathcal{D}_\pi, \mathcal{P}; p_\mathrm{rev})$
  if $\widetilde{\ell} \equiv \ell_\mathrm{r}^{(b)}$ **then** ▷ unchanged instruction
    return $\ell_\mathrm{r}^{(b)}$
  end if
  if $\widetilde{\ell} \in \{\ell_\mathrm{r}^{(1)}, \ldots, \ell_\mathrm{r}^{(k)}\}$ **then**
    return $\ell_\mathrm{r}^{(b)}$
  end if
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{(\ell_\mathrm{r}^{(b)}, \widetilde{\ell})\}$
  $\ell_\mathrm{r}^{(k+1)} \leftarrow \widetilde{\ell}$
end for
return $\ell_\mathrm{r}^{(b)}$

C ROBOMME-FR: ONE-SHOT FAILURE-RECOVERY PROTOCOL

RoboMME-FR is a one-shot physical failure-recovery evaluation protocol built on RoboMME (Dai et al., 2026). RoboMME provides history-dependent manipulation tasks in which the correct high-level decision may depend on past observations, task progress, or previously observed object identities that are no longer evident in the current frame. It also provides released hierarchical VLA implementations with separable high- and low-level policy interfaces, which allows recovery methods to query both levels of the target policy and to evaluate post-failure behavior under a controlled context reset. These properties make RoboMME suitable for studying high-level failure recovery, although RoboMME itself evaluates nominal task completion rather than post-failure recovery.

RoboMME-FR therefore reuses the RoboMME task definitions, success predicates, and target-policy interface, and adds a recovery protocol: nominal failures are saved as shared physical restart states, competing methods generate a recovery instruction offline, and each method may execute only one instruction from that state without resetting the environment, while the target-policy context is reinitialized. This design enables fair comparison of recovery methods under identical failure conditions and isolates instruction quality from environment reset artifacts. The protocol is not specific to RoboMME and is conceptually applicable to other history-dependent manipulation benchmarks that provide compatible hierarchical policy interfaces and controlled post-failure restart states. The remainder of this section details task selection, the evaluation suite, and task-specific recovery semantics.

C.1 TASK SELECTION

We select RoboMME (Dai et al., 2026) tasks according to criteria fixed independently of the performance of POLICYGROUND:

1. the correct high-level decision depends on information from the preceding episode;

Table 3: RoboMME-FR evaluation task suite. Failures are written as $F/50$, where F is the number of nominal-policy failures among 50 episodes.

Task	Description	Representative failure	Failures
ButtonUnmask (BU)	Press a button, then pick containers that hide specified cube colors, in order.	Wrong container, or pickup before the button press	$38/50$
VideoUnmask (VU)	From a setup video, pick containers that hide specified cube colors.	Wrong container after the video-specified target is revealed	$4/50$
ButtonUnmaskSwap (BUS)	Press both buttons, then pick the specified containers; containers may swap mid-episode.	Wrong container or order after the swap	$39/50$
VideoUnmaskSwap (VUS)	From a setup video, pick the specified containers; containers may swap mid-episode.	Wrong swapped container after unmasking	$36/50$
BinFill (BF)	Place exact per-color cube counts into a bin, then press the button.	Wrong color, early stop, or excess count	$22/50$
PickHighlight (PH)	Press a button, then pick all cubes highlighted with white areas.	Non-highlighted cube, or pickup before the button press	$39/50$
VideoRepick (VR)	From a setup video, repeat pick-and-place of the demonstrated cube N times, then press the button.	Wrong cube, early button press, or wrong repetition count	$37/50$
VideoPlaceButton (VPB)	From a setup video, place the correct cube on the target tied to a before/after button-press event.	Wrong cube or wrong event-associated target	$23/50$
VideoPlaceOrder (VPO)	From a setup video, place the correct cube on the ordinal demonstrated target.	Wrong cube or wrong ordinal target	$30/50$
PickXtimes (PX)	Repeat pick-and-place of a specified color onto a target N times, then press the button.	Wrong cube, early button press, or wrong repetition count	$5/50$

2. the target policy produces an identifiable language-level subtask;
3. execution of an incorrect subtask can leave the environment in a potentially recoverable state without resetting the episode; and
4. success can be evaluated using the original task predicate.

We further retain tasks whose post-failure states typically admit a meaningful one-shot recovery instruction (reselecting the intended object, target, or event, or completing a remaining count). The evaluation suite is BUTTONUNMASK (BU), VIDEOUNMASK (VU), BUTTONUNMASKSWAP (BUS), VIDEOUNMASKSWAP (VUS), BINFILL (BF), PICKHIGHLIGHT (PH), VIDEOREPICK (VR), VIDEOPLACEBUTTON (VPB), VIDEOPLACEORDER (VPO), and PICKXTIMES (PX). We exclude STOPCUBE, MOVECUBE, SWINGXTIMES, INSERTPEG, PATTERNLOCK, and ROUTESTICK, whose typical failures do not leave a meaningful recoverable state or do not yield annotatable recovery subtasks under the one-shot protocol.

Table 3 summarizes the suite. Each task contributes 50 nominal-policy episodes (500 in total). Recovery is evaluated only on the nominal failures listed as $F/50$ in the table; these counts match the denominators used in the main results. The suite spans target-selection, identity, counting, and event-/order-tracking failures. Following the default RoboMME evaluation setting, we evaluate 50 episodes per task using the default episode-generation configuration.

One-shot physical recovery protocol. For each selected task, we first run the target policy without recovery. Whenever the policy fails, we save the resulting environment state and retain the complete episode history for the recovery method. Each method is evaluated from the same physical post-

Table 4: Task-specific recovery semantics for the RoboMME-FR evaluation suite (abbreviations as in Table 3). R: recoverable; IR: irrecoverable.

Task	Initial failure type and trigger	Recovery success	Recovery failure
BU	R `premature_bin_pickup`: container lifted before the button press; R `wrong_bin_pickup`: wrong container in the required pickup sequence.	Put down the premature/wrong container, press if still required, then finish the required pickup(s).	Another incompatible container pickup after clear, or timeout.
VU	R `wrong_bin_pickup`: wrong container lifted.	Put down the wrong container, then finish the video-specified pickup(s).	Another incompatible container failure after clear, or timeout.
BUS	R `premature_bin_pickup`: container lifted before both buttons are pressed; R `wrong_bin_pickup`: wrong container in the required pickup sequence.	Put down the premature/wrong container, finish remaining presses, then complete the swap pickup sequence.	Another incompatible container pickup after clear, or timeout.
VUS	R `wrong_bin_pickup`: wrong container lifted.	Put down the wrong container, then finish the video-specified swap sequence.	Another incompatible container failure after clear, or timeout.
BF	R `binfill_insufficient_count`: button pressed before per-color counts are met; IR `binfill_excess_count`: any color exceeds its target;	For R, put in exactly the missing cubes and press the button after counts match.	Any IR failure, another early button press, or timeout.
PH	R `premature_highlight_pickup`: cube lifted before the button press; R `wrong_highlight_pickup`: non-highlighted cube lifted.	Put down the premature/wrong cube, press if still required, then finish highlighted pickups.	Another incompatible wrong-cube pickup after clear, or timeout.
VR	R `invalid_repick_action`: wrong cube or early button press; R `cube_pickup_during_finish`: cube lifted in the final press phase.	Clear the invalid action, then finish remaining pick-and-place repetitions and press.	Another incompatible invalid action after clear, or timeout.
VPB	R `video_place_wrong_target`: wrong cube, or correct cube on the wrong target.	Release/relocate the misplaced cube, then complete the video-specified pick and place.	Another incompatible wrong pickup/target after clear, or timeout.
VPO	R `wrong_video_place_order_pickup`: non-target cube lifted; R `wrong_video_place_order_target`: drop on a non-target.	Release or relocate, then complete the required pickup and correct-target placement.	Another incompatible wrong pickup/target after clear, or timeout.
PX	R `invalid_pickxtimes_action`: wrong cube or early button press; R `cube_pickup_during_finish`: cube lifted in the final press phase.	Clear the invalid action, then finish remaining pick-and-place repetitions and press.	Another incompatible invalid action after clear, or timeout.

failure state, while the target-policy context is reinitialized: the policy receives only the generated recovery instruction and post-restart observations; the pre-failure symbolic history and any original setup demonstration are not re-supplied. The retained episode remains available for diagnosis and proxy-context retrieval, but is not restored as the policy's execution history. We apply recovery to all failed episodes that satisfy the predefined protocol, rather than manually selecting favorable cases.

A method may issue multiple offline black-box policy queries while evaluating candidates. After candidate selection, however, only one recovery instruction is physically executed from the failure state, without resetting the environment. We therefore refer to the protocol as *one-shot physical recovery*, despite the internal iterative probing performed by POLICYGROUND.

Failure detection is provided by the simulator or benchmark evaluator and is not itself evaluated in this work. The recovery horizon is H_{rec}=1300 environment steps (Table 5). An episode is labeled as a timeout if the original task predicate has not been satisfied within this horizon.

C.2 TASK-SPECIFIC RECOVERY SEMANTICS

Table 4 summarizes task-specific failure types and recovery semantics for the ten evaluation tasks. The initial-failure column lists the RoboMME failure conditions that trigger recovery, together with their recoverable (R) or irrecoverable (IR) labels. The recovery-success column describes the continuation required to satisfy the original task predicate, while the recovery-failure column lists incompatible failures that terminate the recovery attempt.

Table 5: Implementation settings used in all experiments.

Setting	Value
High-level policy π_{H}	Qwen3-VL-4B (MME-VLA GroundSG+QwenVL (Dai et al., 2026))
Low-level policy π_{L}	$\pi_{0.5}$ (MME-VLA GroundSG+QwenVL (Dai et al., 2026))
Supervisor F_{SV}	GPT-5.6 Luna (`gpt-5.6-luna`)
Supervisor decoding	API default temperature (unset); seed 7; vision detail `low`
Supervisor images	256×256; global 24 / local 8 / setup 10 frames
Candidate budget K	3
CoP subtask horizon N	4
Action-chain horizon M	8
Action-chunk length	16 (π_{L} open-loop horizon)
Settle radius δ_{settle}	20 mm (start-centered EE enclosing sphere)
Retrieval threshold δ_{ret}	50 mm EE residual
π_{H} sampling	temperature 0; max new tokens 128;
S_{L} trajectory input	workspace Δ-vectors (RDP tolerance 50 mm); front EE overlay
Physical recovery horizon H_{rec}	1300 env. steps (same budget as RoboMME evaluation)
Robot / FK	Franka Emika Panda (ManiSkill `panda_v3` URDF)
Hardware	$2\times$ Intel Xeon Platinum 8558, 3 TiB RAM; NVIDIA H200 (140 GB)

D Experimental Protocol

D.1 Implementation Details

Table 5 lists the frozen models and hyperparameters used in all experiments. Latency is measured as wall-clock time from invocation of the recovery method until the final recovery instruction is returned (Table 5 hardware). Independent supervisor calls may run concurrently (e.g., per-subtask and whole-episode diagnosis; S_{H} and S_{L}); environment reset and physical recovery execution are excluded. We report task-level means together with the mean over all recovery episodes pooled across tasks. Thus, tasks contribute to Avg. in proportion to their number of nominal-policy failures.

D.2 Target-Policy Interface Description

Initial recovery-task generation and instruction revision receive a fixed target-policy interface description $\mathcal{D}_\pi$. It is taken from the RoboMME (Dai et al., 2026) task definitions used by the target policy and contains no episode-specific recovery solution. Concretely, $\mathcal{D}_\pi$ is assembled from three blocks that are handed to p_{init} and p_{rev} for every episode of a task family:

- **General task context**: RoboMME success and failure criteria for the family (and any family-specific operational notes, e.g., opaque bin contents, mid-episode swaps, or "do not press the button"), concatenated with the shared workspace-frame description below.
- **Task templates**: the environment's own full-instruction phrasings for the family (with placeholders such as `<color>`, `<n>`, `<ordinal>`, `<before_or_after>`). Revision is encouraged to stay close to the shortest template that covers every remaining condition in $g^\star$.
- **Subtask templates**: the policy's executable unit-level phrasings. Grounded units typically expose an image slot `<y, x>` that must be estimated from the restart-state frame; the supervisor may adapt wording but is instructed not to invent primitives outside this inventory.

Shared workspace frame. All ten evaluation tasks share the same table-tied millimetre frame (aligned with the front camera; $z{=}0$ is the table surface). The supervisor is told it stands at the near edge looking across at the robot, and that prose left/right/near/far follow this map (the mirror of the robot's own left/right):

- `+x`: toward the near edge of the table (bottom of the front image)
- `−x`: toward the far side of the table (top of the front image)
- `+y`: to the right (right in the front image)
- `−y`: to the left (left in the front image)

Table 6: Target-policy interface description $\mathcal{D}_\pi$ provided during initial generation and instruction revision. General task context is the family-level success/failure prose; task templates are RoboMME full-instruction phrasings; subtask templates are the executable unit inventory (image grounds <y, x> unless noted). The shared workspace frame above is appended to every general task context.

Task	General task context	Task / subtask templates
BU	Success: press the button and pick up all specified containers at least once, in order. Failure: pick any container before the button-press phase, or pick a container hiding a non-specified cube.	*Task:* first press the button on the table, then pick up the container hiding the <color_0> cube[, finally pick up another container hiding the <color_1> cube]. *Subtask:* press the button at <y, x>; pick up the container at <y, x> that hides the <color> cube; put down the container.
BUS	Same as ButtonUnmask, but both buttons must be pressed; some containers may be swapped mid-episode.	*Task:* first press both buttons on the table, then pick up the container hiding the <color_0> cube[, finally pick up another container hiding the <color_1> cube]. *Subtask:* same as ButtonUnmask.
VU	Success: pick up all specified containers at least once, in order when applicable. Failure: pick any container hiding a non-specified cube.	*Task:* watch the video carefully, then pick up the container hiding the <color_0> cube[, finally pick up another container hiding the <color_1> cube]. *Subtask:* pick up the container at <y, x> that hides the <color> cube; put down the container.
VUS	Same as VideoUnmask; some containers may be swapped mid-episode.	Same templates as VideoUnmask.
BF	Success: press the stop button only after the exact required number of cubes for each specified color has been placed in the bin. Failure: fewer or more cubes than required for any color; exceeding a color count terminates early. A release above the bin counts as put-in; contents are not visually verifiable, so track executed pickups/put-ins. If the button was pressed early, put in exactly the missing cubes and press again.	*Task:* put the cubes into the bin, then/and press the button to stop; put <n_0> <color_0> cube(s) [, and <n_1> <color_1> cube(s) [, and <n_2> <color_2> cube(s)]] into the bin, then/and press the button to stop (0–3 color variants). *Subtask:* pick up the <color> cube at <y, x>; put it into the bin at <y, x>; press the button to stop at <y, x>.
PH	Success: press the button and pick up every cube highlighted with a white area at least once. Failure: attempt a pick before the button press, or pick a non-highlighted cube.	*Task:* first press the button, then pick up all cubes that have been highlighted with white areas on the table; first press the button, then pick up all highlighted cubes, finally press the button again to stop. *Subtask:* press the button at <y, x>; pick up the highlighted cube at <y, x>, which is <color>; pick up the <ordinal> highlighted cube at <y, x>, which is <color>; place the cube onto the table.
VR	Success: complete exactly N pick-and-place repetitions with the demonstrated cube, then press the button. Failure: pick the wrong cube; press the button before finishing N repetitions; or complete more than N repetitions.	*Task:* watch the video carefully, then repeatedly pick up and put down the same block that was previously picked up for <n> times, finally put it down and press the button to stop (and shorter <n>-times variants). *Subtask:* pick up the correct cube at <y, x> for the <ordinal> time; put it down; press the button to finish.
VPB	Success: pick the correct color cube and place it on the correct target (purple disks). Failure: wrong cube or wrong target. Do not press the button in this task.	*Task:* watch the video carefully, then place the <color> cube on the target right / where it was placed <before_or_after> the button was pressed (and last-before / first-after specializations). *Subtask:* pick up the <color> cube at <y, x>; place the cube on the target at <y, x>.
VPO	Success: pick the correct colored cube and place it on the target indicated by the specified placement order (purple disks). Failure: wrong cube or wrong target. Do not press the button in this task.	*Task:* watch the video carefully, then place the <color> cube on the <ordinal> target it was previously placed on (and "where it was placed" variant). *Subtask:* pick up the <color> cube at <y, x>; place the cube on the target at <y, x>.
PX	Success: complete exactly the specified number of pick-and-place repetitions, then press the button (target is a purple disk). Failure: pick a wrong cube; press the button before all repetitions; or perform more repetitions than specified.	*Task:* pick up the <color> cube and place it on the target, repeating this [pick-and-place] action <n> times, then press the button to stop. *Subtask:* pick up the <color> cube for the <ordinal> time; place the <color> cube onto the target; press the button to stop.

- +z: up, away from the table
- −z: down, toward the table surface

Per-task specifications. Table 6 lists, for each evaluation task, the general task context together with the concrete task and subtask templates that constitute $\mathcal{D}_\pi$. Placeholders are filled from the (recovery) goal at generation / revision time; <y, x> is re-estimated from the restart-state observation rather than copied from the failed rollout.

D.3 Metrics and Statistical Analysis

Let N_t be the number of original evaluation episodes for task t, and let F_t be the number of episodes on which the target policy fails before recovery. Metrics restricted to post-failure recovery attempts use F_t as their denominator, whereas nominal and overall task-success rates use N_t.

- **Original success rate**: the fraction of the N_t episodes completed by the target policy without recovery.
- **Recovery success rate (RS)**: the fraction of the F_t failed episodes for which recovery satisfies the original task-success predicate.
- **High-level alignment rate (HA)**: the fraction of the F_t failed episodes whose post-resume critical-path checkpoint sequence is a valid prefix of the remaining oracle plan derived from the resume snapshot (and optional recovery goal). Episodes that emit no checkpoint, including diagnosis failures, missing initial candidates, and skipped recovery executions, are counted as misaligned. Alignment is scored independently of physical success or timeout; see Appendix D.4.
- **Overall success rate**: the fraction of the original N_t episodes completed either without recovery or through a successful recovery attempt.
- **Recovery latency**: the wall-clock time from invoking the recovery method to producing the final recovery instruction, excluding subsequent physical execution.
- **Query cost**: the numbers of supervisor-LMM, high-level-policy, and low-level-policy queries required to produce the final recovery instruction.

Because tasks contain different numbers of nominal-policy failures, aggregate recovery metrics are reported as micro-averages over all failed episodes. Specifically, for any binary recovery metric with task-level numerator X_t,

$$\text{Avg.} = \frac{\sum_t X_t}{\sum_t F_t}.$$

Thus, RS and HA use the same pooled denominator $\sum_t F_t = 273$. For episode-level continuous quantities such as recovery latency, Avg. is the mean over all recovery episodes pooled across tasks. We additionally report task-level rates together with the number of nominal-policy failures for each task.

D.4 High-Level Alignment

Procedure. For each nominal-policy failure episode included in RoboMME-FR, we score the high-level policy from the resume state until the terminal recovery outcome ($\mathrm{success}$, failed, or $\mathrm{timeout}$). Non-checkpoint subgoals (e.g., approach, put-down on VideoRepick) are ignored. Within-step repeats and tight pick/put oscillations with small coordinate jitter are collapsed before matching; compound "and"/"then" phrases are split after collapse.

Let $P = (p_1, \ldots, p_{|P|})$ be the resulting checkpoint sequence and $E = (e_1, \ldots, e_{|E|})$ the expected remaining critical-path checkpoints from the resume snapshot. We require E to be defined for every evaluation episode; failure to construct E is treated as an evaluation-system error rather than a metric outcome. The episode is

$$\begin{cases} \text{aligned}, & 1 \le |P| \le |E| \text{ and } p_i \sim e_i \text{ for all } i \le |P|, \\ \text{misaligned}, & \text{otherwise}. \end{cases} \tag{7}$$

Here $p \sim e$ denotes the task-specific match below (action/entity, and coordinates when grounding is required). In particular, an empty checkpoint sequence ($|P| = 0$) is counted as misaligned. This includes episodes in which semantic recovery specification fails to produce an initial recovery instruction or recovery execution is otherwise skipped.

With A_t aligned episodes among the F_t nominal-policy failures,

$$\text{HA}_t = \frac{A_t}{F_t}. \tag{8}$$

Physical success does *not* imply alignment: a recovery may succeed with a wrong checkpoint trace, or fail/timeout while still emitting a valid prefix of E.

Table 7: Checkpoint actions and grounding for HA. "Ordered" means remaining targets are matched in sequence; "unordered" allows any remaining candidate.

Task	Ground	Critical-path checkpoints
VU / VUS	yes	ordered remaining container `pick_up`
BU / BUS	yes	optional leading `press`, then ordered container `pick_up`
PH	yes	optional `press`, then unordered remaining highlighted `pick_up`
VR	yes	remaining demonstrated-cube `pick_up` (puts ignored)
VPB / VPO	yes	`pick_up` then `place` (holding at resume skips pick)
BF	no	remaining color `pick_up`/`put_into`, then `press`
PX	yes	remaining `pick_up`/`place` cycles, then `press`

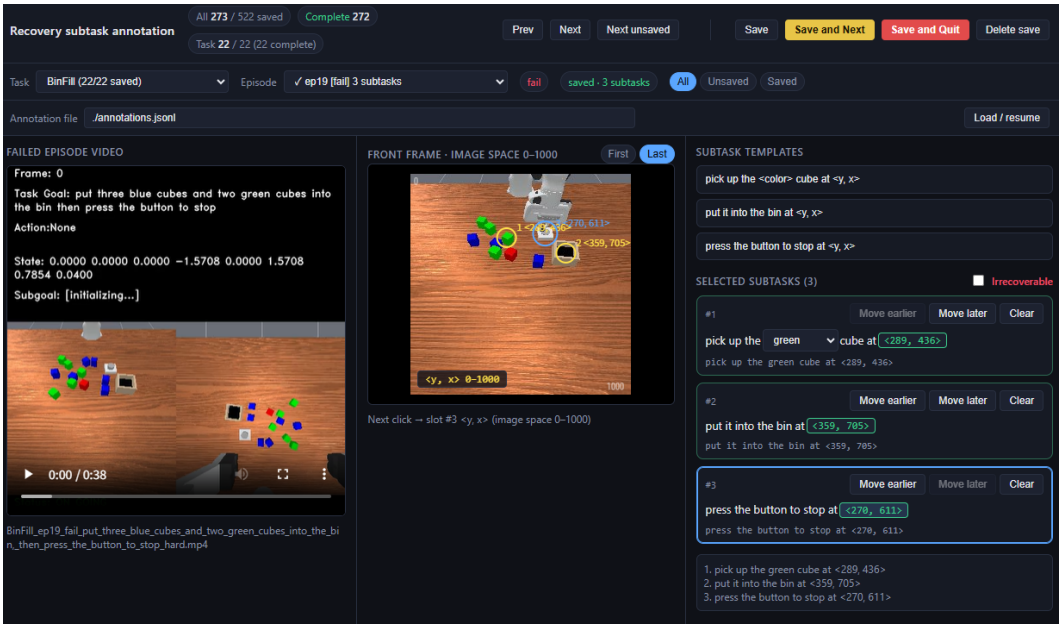


Figure 4: Interface used to construct the human-instruction reference. The annotator inspects the failed rollout, task instruction, execution history, and final observation, and composes a recovery instruction from task-specific subtask templates. The resulting instruction is subsequently executed by the same frozen target policy from the saved post-failure state.

Grounding. When a step requires grounding, policy and expected coordinates must lie within a fixed pixel tolerance (default 20 px in the policy image frame). Expected locations are taken from the ground-truth grounded subgoal associated with the resume state, expressed in the policy image frame, rather than from free-form coordinates in the generated recovery goal. BinFill is the exception: only color/count semantics are matched (no coordinate check).

Task-specific checkpoints. Table 7 summarizes which emitted actions count as checkpoints and whether coordinates are required.

Briefly: Unmask families match the next container color (and location) from the resume filled segment, then later goal colors without forcing unavailable coordinates. PickHighlight matches any remaining highlighted cube by ground (color optional). VideoRepick counts only picks toward the filled cube location. VideoPlaceOrder and VideoPlaceButton require a grounded place onto the target (and a grounded pick unless already holding). BinFill matches remaining per-color pick→bin put counts without coordinates. PickXtimes matches remaining pick/place cycles against filled pick or place coordinates when present, then the stop press.

Table 8: Task-level ablations of policy-grounded recovery (detail of Table 2). RS measures physical recovery; HA measures alignment of the executed high-level checkpoint sequence. H/L indicate high-/low-level feedback. "Failed eps." gives the number of nominal-policy failures. "Avg." denotes the micro-average over all failed episodes. "Ref." marks the human instruction reference row.

			RS (%)										
Variant	H	L	VU	BU	VUS	BUS	PH	VR	VPB	VPO	BF	PX	Avg.
Failed eps.			4	38	36	39	39	37	23	30	22	5	–
Semantic recovery			0.0	10.5	5.6	10.3	2.6	2.7	8.7	10.0	18.2	0.0	7.7
H only	✓		0.0	18.4	11.1	12.8	0.0	0.0	39.1	16.7	22.7	20.0	13.2
L only		✓	50.0	42.1	36.1	20.5	28.2	2.7	30.4	13.3	27.3	0.0	24.9
PolicyGround	✓	✓	50.0	47.4	41.7	12.8	20.5	2.7	17.4	16.7	27.3	0.0	23.4
Human instruction	Ref.		50.0	47.4	47.2	25.6	28.2	13.5	47.8	13.3	27.3	0.0	30.8
			HA (%)										
Variant	H	L	VU	BU	VUS	BUS	PH	VR	VPB	VPO	BF	PX	Avg.
Semantic recovery			100.0	94.7	75.0	48.7	79.5	29.7	34.8	43.3	59.1	80.0	60.8
H only	✓		100.0	97.4	72.2	51.3	71.8	37.8	60.9	50.0	50.0	100.0	63.7
L only		✓	100.0	100.0	69.4	43.6	74.4	32.4	34.8	33.3	45.5	60.0	57.1
PolicyGround	✓	✓	100.0	92.1	75.0	41.0	79.5	29.7	39.1	53.3	54.5	60.0	60.1
Human instruction	Ref.		100.0	97.4	100.0	94.9	100.0	100.0	87.0	80.0	100.0	80.0	95.2

D.5 Human-Instruction Annotation Protocol

To construct the human-instruction reference, we developed the annotation interface shown in Figure 4. For each failed episode, the annotator is provided with the original task instruction, the failed rollout video, the subtask and robot-state history, and the final observation at which recovery begins. The annotator then constructs a recovery instruction intended to complete the original task from this post-failure state.

To keep the reference compatible with the action repertoire of the frozen hierarchical policy, the interface provides task-specific subtask templates and their arguments, including object attributes and image-space locations. The annotator may compose and order multiple subtasks, for example, first placing a currently held object on the table and then selecting the correct target. Episodes judged irrecoverable using the available subtask repertoire can be explicitly marked as such.

Annotations are created before recovery execution. The annotator does not observe the outputs, probe scores, or execution outcomes of PolicyGround. Each completed human instruction is evaluated from the same saved physical post-failure state, with the same context-reset target-policy interface and recovery horizon as the automatic methods. The target policy receives the human-written recovery instruction but not the pre-failure policy history or setup demonstration. We therefore use this condition as an empirical human-specified reference for recovery-instruction construction, not as a theoretical upper bound on recovery performance.

E Detailed Experimental Results

E.1 Full Ablation Results

Table 8 expands Table 2 with task-level RS and HA for the hierarchy-feedback variants. The results reveal substantial task-dependent heterogeneity rather than uniformly additive gains from the two feedback sources.

At the aggregate level, the variants form an RS–HA trade-off. High-level feedback alone achieves the highest automatic HA (63.7%) but a lower RS of 13.2%, whereas low-level feedback alone achieves the highest RS (24.9%) while reducing HA to 57.1%. The full method lies between these specialized variants, achieving 23.4% RS and 60.1% HA. Relative to low-level feedback alone, combining both feedback sources recovers 3.0 points of HA at a cost of 1.5 points of RS; relative to high-level feedback alone, it gains 10.2 points of RS at a cost of 3.6 points of HA.

Table 9: Mean recovery-instruction generation latency (s) on ten RoboMME-FR tasks. Lower values are better. H/L indicate high-/low-level feedback. Average (Avg.) is the mean over all recovery episodes pooled across tasks.

Method	VU	BU	VUS	BUS	PH	VR	VPB	VPO	BF	PX	Avg.
Semantic recovery	38.7	32.6	41.8	50.3	39.4	43.2	52.8	54.3	61.9	40.6	45.4
H only	81.3	59.6	76.5	75.3	62.8	95.8	87.9	83.0	97.4	79.3	78.1
L only	96.6	80.4	94.5	91.2	84.0	94.6	89.0	95.2	113.1	79.7	91.5
PolicyGround	114.4	81.6	107.3	104.4	92.9	92.8	94.4	109.3	140.6	93.5	101.0

The preferred feedback source also varies substantially across tasks. Low-level feedback is particularly effective on BUS and PH, whereas high-level feedback performs best on VPB and PX. The full method is strongest on BU and VUS and ties the best automatic result on several other tasks. Thus, neither hierarchy-specific signal uniformly dominates across tasks. This heterogeneity suggests that task- or candidate-adaptive weighting may improve the current fixed score combination.

These ablations are not additive modules applied to a fixed candidate set. Removing one feedback source changes the instruction supplied to the reviser and therefore changes the entire subsequent candidate trajectory. Consequently, combining both feedback sources need not recover the task-wise maximum of the two specialized variants. The observed heterogeneity suggests that task- or candidate-adaptive weighting of hierarchy-specific feedback may improve the current fixed combination.

E.2 Inference-Cost Details

Table 9 reports task-level wall-clock latency for recovery-instruction generation on the hardware in Table 5. Physical recovery execution is excluded.

Query complexity. Let $C \leq K$ be the number of candidates evaluated before convergence, $N^{(k)} \leq N$ the number of subtasks predicted for candidate k, and $J_i^{(k)} \leq M$ the number of low-level action chunks probed for its i-th subtask. The numbers of high- and low-level target-policy queries are

$$Q_{\mathrm{H}} = \sum_{k=1}^{C} N^{(k)} \leq CN, \qquad Q_{\mathrm{L}} = \sum_{k=1}^{C} \sum_{i=1}^{N^{(k)}} J_i^{(k)} \leq CNM. \tag{9}$$

Thus, N and M specify worst-case probe horizons, whereas Q_{H} and Q_{L} count the actual calls after early stopping and convergence.

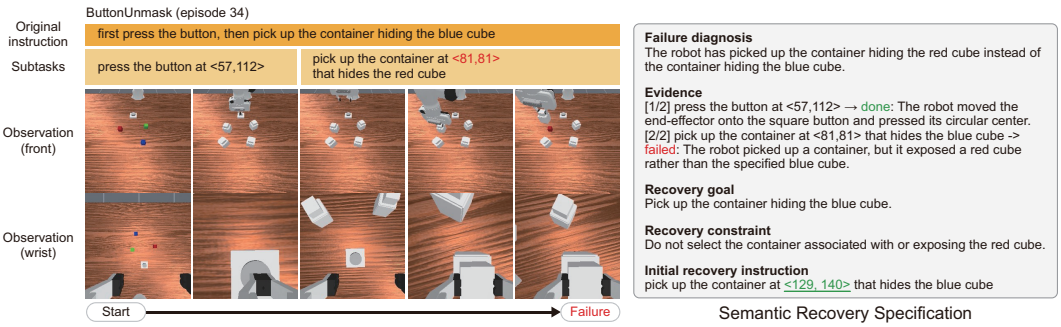
Let Q_0 denote the supervisor calls for semantic specification and initial instruction generation, which depend on the number of subtask intervals in the failed episode, and let $R \leq C - 1$ be the number of revision calls. Let Q_{SV} denote the number of supervisor-LMM calls. Each evaluated candidate requires one high-level evaluation call and one low-level evaluation call per predicted subtask. Therefore,

$$Q_{\mathrm{SV}} = Q_0 + \sum_{k=1}^{C} (1 + N^{(k)}) + R \leq Q_0 + C(1 + N) + (C - 1). \tag{10}$$

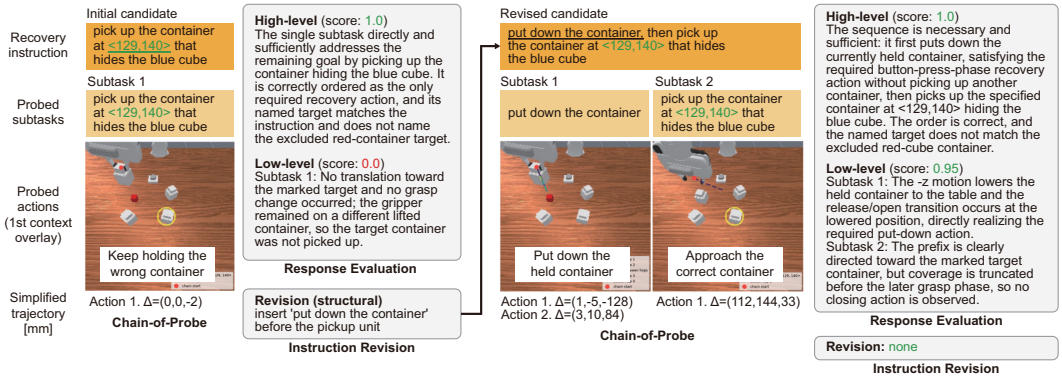
E.3 Qualitative Case Study

Figure 5 traces a representative ButtonUnmask recovery episode through semantic specification, policy-grounded revision, and physical execution. The nominal policy picks up the container hiding the red cube instead of the target container hiding the blue cube. The supervisor correctly diagnoses this failure and identifies the remaining recovery goal, but its initial instruction omits the prerequisite of releasing the incorrectly held container. This episode therefore isolates a candidate-realization failure rather than an error in semantic diagnosis.

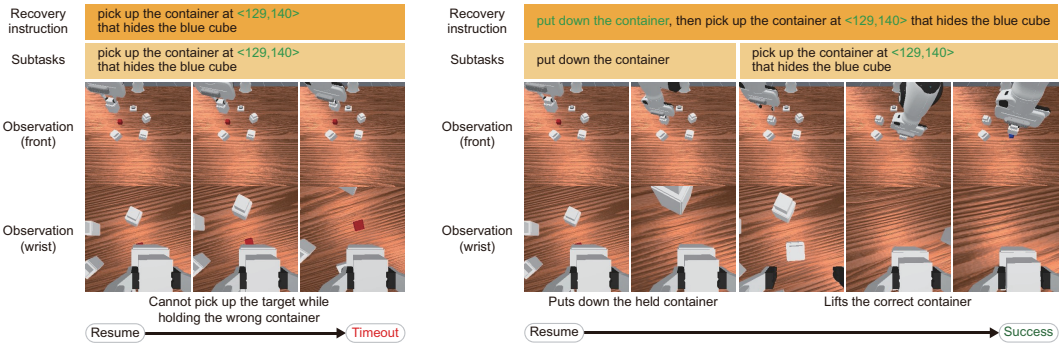
This example highlights the complementary roles of hierarchy-specific feedback. High-level evaluation alone assigns the same maximal score to both candidates because they identify the correct



(a) Failure episode and semantic specification



(b) Policy-grounded revision trace



(c) Physical recovery outcomes

Figure 5: Qualitative case study on ButtonUnmask (episode 34). (a) Semantic specification identifies the correct target, but the initial recovery instruction omits releasing the held non-target container. (b) High-level probing confirms the target, while low-level probing exposes the missing prerequisite. Structural revision inserts a put-down subtask, after which the revised candidate receives compatible policy responses. (c) Physical execution of the initial instruction times out, whereas the policy-grounded instruction releases the held container and successfully picks up the target. Evaluator feedback is abridged for readability.

Table 10: Performance as a function of the revision procedure. Single revision forcibly executes the first revised candidate, whereas Full uses adaptive revision and best-so-far selection.

Variant	Avg. RS ↑	Avg. HA ↑
No revision	7.7	60.8
Single revision	22.7	60.1
Full adaptive search	23.4	60.1

Table 11: Candidate selection and evaluation depth over all 273 recovery episodes. $K = 3$ is the candidate-evaluation budget (Algorithm 3): $k = 1$ is the initial candidate and $k = r$ is the candidate after $r-1$ revisions. The right panel counts candidates evaluated, not revision calls. "No candidate" denotes episodes in which semantic recovery specification does not produce an initial recovery instruction. Fractions use all 273 episodes as the denominator; these episodes are counted as recovery failures for RS and as misaligned for HA.

Final selected candidate			Candidates evaluated		
Index	Episodes	Fraction	Count	Episodes	Fraction
No candidate	5	1.8%	0	5	1.8%
$k = 1$	104	38.1%	1	83	30.4%
$k = 2$	126	46.2%	2	117	42.9%
$k = 3$	38	13.9%	3	68	24.9%
Total	273	100%	Total	273	100%

remaining target. Low-level feedback exposes the missing prerequisite in the initial candidate and provides the evidence needed for structural revision. Thus, the benefit of policy grounding in this case is not a correction to the semantic recovery goal, but a change in how that goal is realized by the target hierarchy.

E.4 Revision-Depth Analysis

We further analyze how recovery performance and candidate selection change with revision depth. The single-revision variant evaluates the initial candidate, performs one policy-guided revision, and forcibly executes the revised candidate without further revision or best-so-far selection. In contrast, the full method may perform multiple revisions and returns the highest-scoring candidate observed during the search. Therefore, their difference reflects both additional refinement and best-so-far candidate selection.

A single revision already captures most of the improvement, achieving 22.7% RS and 60.1% HA, compared with 23.4% RS and 60.1% HA for the full method (Table 10). Thus, the first policy-guided revision is responsible for the dominant performance gain, while deeper search provides only a modest additional improvement in RS.

Table 11 separately reports the candidate ultimately selected by the full method and the number of candidates evaluated under budget $K = 3$. These quantities differ because the method retains the best-so-far candidate: evaluating a revised instruction does not necessarily replace an earlier candidate.

Failure diagnosis fails to produce an initial recovery instruction in 5 of the 273 episodes; these episodes evaluate no candidate and return none. All remaining 268 episodes invoke the reviser at least once. Among them, 69.0% evaluate two or more candidates, and 25.4% exhaust the candidate budget $K = 3$. Nevertheless, the initial or first-revision candidate (k=1 or k=2) is ultimately selected in 85.8% of these episodes, and the second-revision candidate (k=3) in 14.2%. Thus, later candidates are frequently evaluated, but the second revision replaces the earlier best candidate only in a limited subset; with $K = 3$, no further candidate index exists.

Episodes for which failure diagnosis produces no recovery instruction are counted as recovery failures for RS and as misaligned for HA.