RECOVER, DON'T JUST CONTINUE: LEARNING FROM FEEDBACK FOR PHYSICAL FAILURE UNDERSTANDING AND CORRECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Multimodal large language models are increasingly seen as the cognitive foundation for embodied intelligence, yet planning from instructions alone cannot make them reliable embodied brains. During physical execution, actions deviate from intended effects, and what can be realized is jointly constrained by the embodiment, controller, and environment. An embodied brain must therefore understand why execution deviates, recognize what the current execution system can reliably realize, and revise decisions using interaction experience. These three needs correspond to causal failure localization, capability-constrained adaptation, and experience-driven revision, making them complementary and non-substitutable. However, current embodied foundation models, despite advances in spatio-temporal reasoning, physical-world modeling, and high-level planning, largely lack these capabilities. To address this gap, we introduce Embodied-Corrector. Controlled failure–recovery data connects actions, observable consequences, failure causes, and executable recoveries, providing physical causal supervision. Evaluation-first SFT with policy-optimization internalization transfers verified evaluations into the model's own reasoning, so it assesses outcomes and failure causes before inferring recoveries a frozen executor can realize, enabling execution-capability adaptation. A CoT template requires the model to review execution history, evaluate failure causes, and plan recovery, so each execution informs the next decision without parameter updates, realizing in-context learning from interaction. This internalizes diagnostic and corrective experience into the model's reasoning. Experiments on embodied failure reasoning and planning benchmarks show stronger failure-cause analysis and corrective planning; with a frozen VLA, composite task success improves by 6.1%, and with fixed GPT-5.5 execution, embodied manipulation and navigation tasks improve by 13.9% macro-average.

1 INTRODUCTION

Multimodal large language models are becoming an important foundation for embodied intelligence Driess et al. (2023); Brohan et al. (2023a); Ahn et al. (2022a); Reed et al. (2022). By combining visual understanding with language reasoning, they can interpret task goals, reason about scene relations, and decompose complex instructions into executable steps Liang et al. (2023a); Huang et al. (2022a); Singh et al. (2023); Song et al. (2023). Recent embodied foundation models such as RynnBrain Dang et al. (2026a), RoboBrain 2.0 BAAI RoboBrain Team (2026), and Embodied-Reasoner Zhang et al. (2026a) have advanced spatio-temporal memory, physical-world reasoning, and embodied planning; VeBrain-1.5 VeBrain Team (2026) further unifies multimodal perception, spatial reasoning, and robot control in a shared decision interface. However, a reliable embodied brain requires more than planning. During physical execution, actions deviate from intended effects, and what can be realized is jointly constrained by the embodiment, controller, and environment Anonymous (2026d). An embodied brain must therefore understand why execution deviated, recognize what the current executor can reliably realize, and adjust decisions using interaction experience. This raises the central question: *how can a multimodal model evolve from planning what to do into an embodied brain that understands physical interaction, adapts to execution capability, and corrects its behavior after failure?*

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.

We argue that this requires three related but non-substitutable capabilities. First, **physical causal understanding and failure localization** Ying et al. (2026); Liu et al. (2026a); Anonymous (2026a) requires determining not only that an execution failed, but also which action, physical relation, or unmet prerequisite caused it. Existing failure-memory and detection methods can flag unsuccessful outcomes Ying et al. (2026), yet a model may observe an object not placed into a container while failing to distinguish incorrect grasp timing, inappropriate contact pose, or an earlier uncompleted step. Effective correction requires tracing execution back to the condition that first invalidated progress. Second, **execution-capability identification and adaptation** Physical Intelligence Team (2026); Ghosh et al. (2024) requires translating intent into a form the current executor can reliably realize. Even a correct diagnosis is ineffective if the recovery exceeds the controller's capability. The brain should adapt instruction content and granularity, e.g., decomposing a long request into state-conditioned subtasks for a frozen VLA Physical Intelligence Team (2026); Brohan et al. (2023b); O'Neill et al. (2024). Third, **in-context learning from interaction experience** Chen et al. (2026); Zhang et al. (2026c); Sridhar et al. (2025) requires information from one execution to change the next decision. Prior work indicates that a robot's own physical interaction experience can support in-context learning even when the policy model is frozen Chen et al. (2026). Failure observations, previous actions, and successful corrections should become evidence for planning rather than accumulating in context while the model repeats the same behavior Anonymous (2026b;c).

We first examine whether current embodied brains possess these capabilities. On failed EB-ALFRED Yang et al. (2025b) executions, we supply frozen multimodal models with the same task goal, visual observations, executed actions, and execution feedback, and inspect predicted failure cause and proposed next-step correction. As shown in Figure 1(a), models such as RynnBrain Dang et al. (2026a), MiMo-Embodied Xiaomi MiMo Team (2026), and InternVL3 OpenGVLab (2025) frequently produce plausible descriptions while failing to identify the condition that actually blocks progress, and their plans often fail to repair it. This exposes weak physical causal understanding and failure localization. The missing capability is not merely recognizing failure, but transforming execution evidence into an explanation of *why* it failed and *what must change next*.

We next ask what form this missing information should take. Keeping the evaluated brain, visual input, action interface, and decoding fixed on EB-ALFRED, we vary only the feedback after failure. As shown in Figure 1(b), repeating the failed outcome provides little benefit Shinn et al. (2023a), whereas supplying the concrete failure cause consistently improves performance. For RoboBrain2.5-4B, MiMo-Embodied-7B, and InternVL3-8B, cause-level feedback changes task success from 0.22 to 0.28, from 0.20 to 0.27, and from 0.20 to 0.24. Useful feedback does not simply state failure; it identifies which expected condition was violated and why that prevents progress, revising the brain's interpretation of the task state. Together, these diagnostics indicate that current models lack not only failure analysis, but also execution-capability adaptation and in-context learning from interaction: outcome repetition alone does not change subsequent decisions, and only explicit causal feedback enables correction.

Motivated by this, we introduce **Embodied-Corrector**, which couples the three capabilities through data, training, and inference. For *physical causal understanding and failure localization*, we build controlled failure–recovery data by branching successful trajectories at subtask boundaries and intervening on navigation, reachability, grasp pose, control timing, articulation state, task prerequisites, and target grounding. Each branch shares the preceding successful context while changing one controlled factor, and we retain only failures observable from public execution evidence and recoverable from the realized state. The resulting paired failure–recovery experience connects what changed, what consequence it caused, and what must be restored.

To obtain this data at scale, we build an automated pipeline coordinating simulation environments, robot execution interfaces, multimodal models, and auxiliary tools for trajectory generation, failure construction, cause annotation, and correction verification; a Harness provides runtime support for interaction and data collection. For *execution-capability identification and adaptation*, evaluation-first SFT first learns Judge and Cause, then Sub and Fine, while policy-optimization internalization transfers verified evaluations into the model's own reasoning distribution; deployment therefore relies on the model's own assessment to generate state-conditioned recoveries that a fixed executor can realize. For *in-context learning from interaction experience*, a CoT template requires the model to review execution history, evaluate the failure cause, and plan the recovery, so it conditions the next decision on previous interaction evidence without parameter updates. This internalizes diagnostic and corrective experience into the model's reasoning, enabling better decisions in new executions.

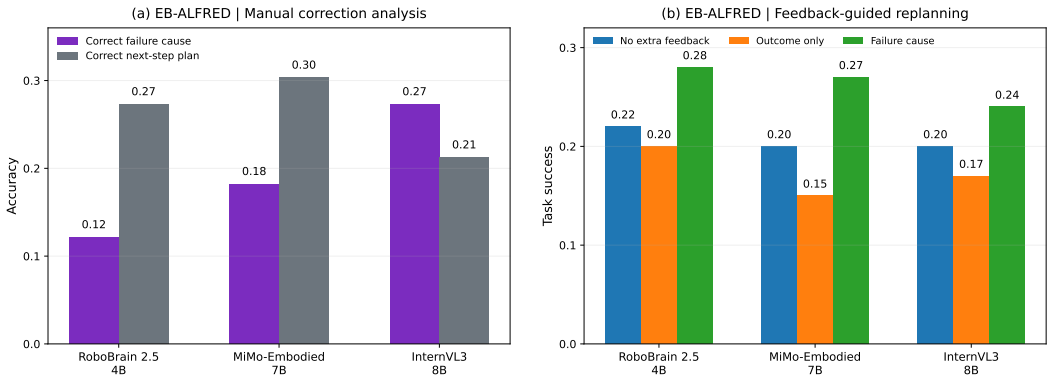

Figure 1: Diagnosing missing capabilities and feedback effects. (a) Failure-cause identification. (b) Cause-level feedback helps; repeated outcome does not.

Experiments show these three capabilities are complementary and individually necessary. First, on brain-level corrective reasoning, Embodied-Corrector substantially improves failure-cause analysis and produces recovery plans consistent with the identified physical cause, demonstrating stronger physical causal reasoning rather than outcome classification alone. Second, with the VLA frozen, execution-aware decomposition improves $\pi_{0.5}$ by **5.4%** on Composite-Seen tasks and by **6.1%** on Composite-Unseen tasks, showing that the brain can adapt task granularity to reduce the burden on a fixed executor. Third, with the GPT-5.5 code generator, robot functions, perception module, and execution budget all fixed, feeding Embodied-Corrector's diagnosis and recovery back into the next planning context improves all ten execution conditions over the untrained corrective brain, with a macro-average gain of **13.9%**. This shows that execution feedback is not merely stored in history: it changes the next workflow produced by the same planner. Our contributions are threefold:

- We identify physical causal understanding, execution-capability adaptation, and in-context learning from interaction feedback as three complementary capabilities for corrective embodied reasoning, and diagnose their absence in current multimodal embodied brains.
- We introduce Embodied-Corrector and a controlled failure–recovery data construction pipeline connecting actions, observable physical consequences, failure causes, and executable recoveries, together with evaluation-first SFT.
- We evaluate correction from brain-level failure reasoning to fixed-backend execution, showing the learned brain can explain failures, adapt instructions to a frozen VLA, and provide corrective context that changes the subsequent thinking of a code generator.

2 Failure Data and Corrective Brain Training

Recent code-based embodied systems improve robot-function invocation, skill composition, and routing to fixed execution backends, while execution-aware frameworks provide trajectory feedback for diagnosis and recovery (Fu et al., 2026a; Ding et al., 2026; Pan et al., 2026). However, they provide limited supervision for task-level failure understanding: models are rarely trained to judge whether a subtask succeeded, identify the broken observable prerequisite, and decide whether to advance or recover. We therefore construct controlled failure trajectories from successful demonstrations and convert them into supervision for outcome judgment, cause diagnosis, recovery-subtask selection, and fine-grained planning. Because evaluation and planning are difficult to learn jointly, we use an evaluation-first curriculum and finally apply on-policy distillation (OPD) to internalize verified evaluation into the planning-token distribution.

2.1 Dataset and Benchmark Construction

Controlled failure trajectories. We begin with successful RoboCasa365 and OmniBehavior trajectories (Nasiriany et al., 2026; Li et al., 2024). Subtask boundaries are identified from task-graph

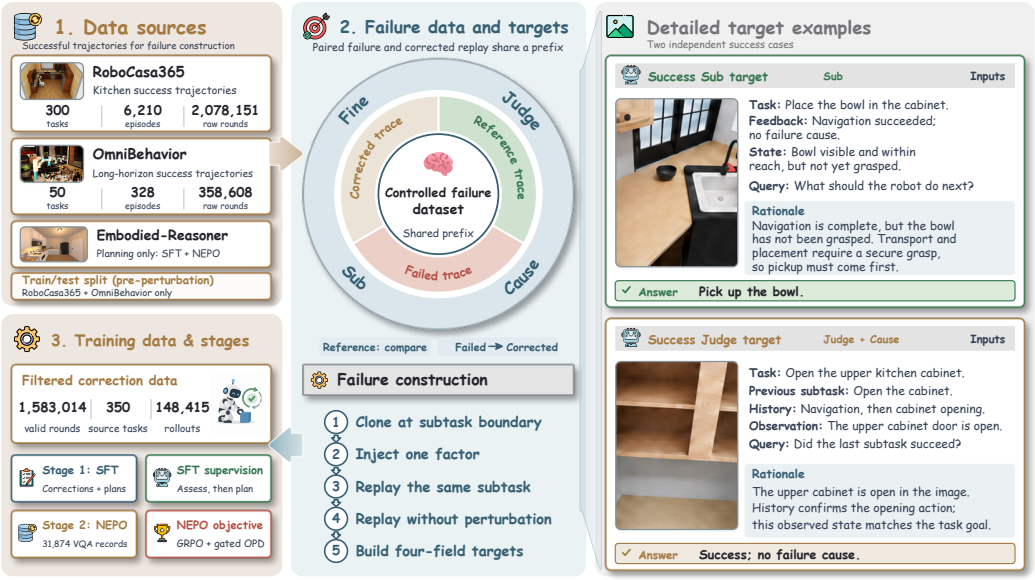

Figure 2: Controlled failure-data construction. Successful trajectories are branched through controlled perturbations and replayed to produce aligned supervision for outcome judgment, cause diagnosis, recovery-subtask selection, and fine-grained planning.

transitions and task-relevant state changes, such as reaching a target, grasping an object, or opening a container. At each boundary, we retain the task goal g_i, current subtask, executed workflow w_i, pre-execution observation o_i, and corresponding successful segment. We then clone the simulator state before the subtask, inject one known perturbation, and execute the same subtask again. Each branch changes only one factor, making its failure source identifiable from the intervention. Perturbations cover navigation, reachability, grasp pose, control timing, articulation, task prerequisites, and target grounding. The replay produces a post-execution observation o_{i+1} and execution trace τ_i, including function arguments, returns, exceptions, and key visual frames.

Correction-target construction. Judge and Cause are generated from the known perturbation and intervention point. Judge records whether the perturbed execution completes the current subtask. Cause combines the perturbation with its realized consequence and describes the failure using publicly observable evidence. For example, a changed grasp height that leaves the object unattached is written as "the gripper closed above the target and the object was not grasped," rather than exposing a hidden simulator parameter. For a failed example, the current subtask remains incomplete and is used as Sub. For a successful example, Sub is the next subtask in the original trajectory. Thus, Sub measures whether the model can decide to advance or repeat the interrupted step. Fine specifies how to execute Sub. We remove the perturbation and replay the same subtask to obtain a corrected segment aligned with the failed trajectory. A fixed closed-source model summarizes the failed and corrected traces into an ordered fine-grained plan while preserving the target object, motion direction, end-effector state, action order, and verification steps. Fine is therefore a language description of the corrected trajectory rather than a template selected from the perturbation type.

Filtering and benchmark construction. We discard ineffective perturbations, unsynchronized media, inconsistent object identities, invalid traces, and failures that cannot be identified from observations. Source trajectories are split before perturbation generation. The corrective training set contains aligned Judge, Cause, Sub, and Fine targets from training trajectories. **EmbodiedDiag-Bench** is generated only from held-out trajectories and therefore shares neither source trajectories nor successful prefixes with training. We report each field separately, while the joint score measures whether the model can transform execution evidence into a corrected task state and recovery decision.

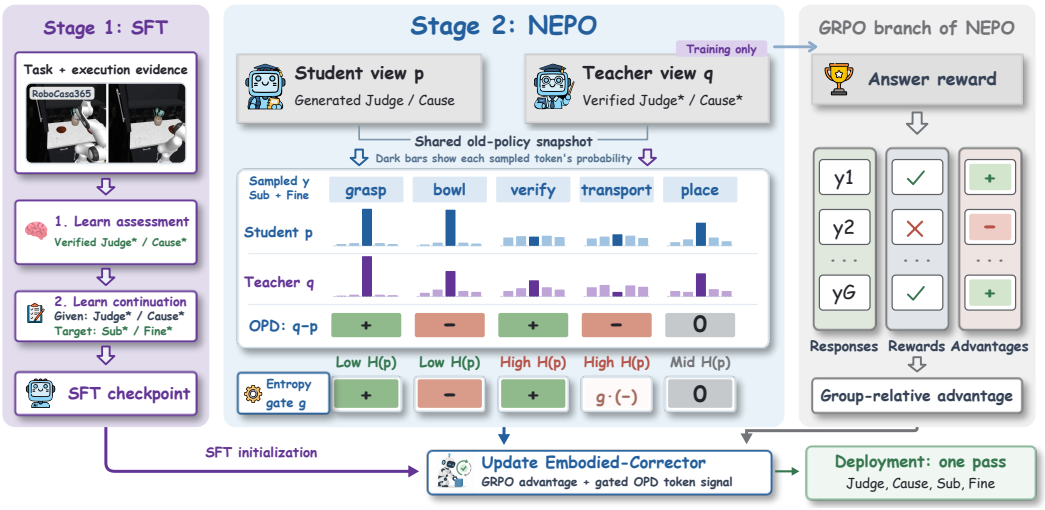

Figure 3: Overview of Embodied-Corrector training. Training proceeds from execution evaluation to recovery planning, followed by NEPO optimization that reduces the mismatch between verified evaluation prefixes used during training and model-generated prefixes used at deployment.

2.2 Evaluation-First Brain Training and OPD Internalization

Embodied-Corrector is a unified multimodal VQA model that autoregressively generates Judge, Cause, Sub, and Fine. An incorrect evaluation prefix can distort the recovery plan, while a response-level reward is too coarse to identify the responsible planning token. We therefore first train evaluation, then recovery planning, and finally use OPD to internalize verified evaluation. Deployment still uses one inference to generate the complete response.

Stage 1: Evaluation-first supervised training. We first train Judge and Cause, requiring the model to determine whether the subtask was completed and explain its observable failure cause. We then train Sub and Fine conditioned on verified Judge and Cause. This curriculum teaches the dependency from execution outcome to cause diagnosis and recovery planning without requiring difficult joint prediction from the beginning.

Stage 2: OPD-based evaluation internalization. Teacher-forced training conditions Sub and Fine on verified evaluations, whereas deployment uses the model's own Judge and Cause. This mismatch allows an evaluation error to propagate through later planning tokens. We therefore use OPD to transfer information from the verified evaluation into the student's planning distribution.

Let $a_i^* = (d_i^*, c_i^*)$ denote the verified evaluation, $\hat{a}_i = (\hat{d}_i, \hat{c}_i)$ the student-generated prefix, and $y_i = (y_{i,1}, \ldots, y_{i,T})$ the planning tokens. Using the same old-policy snapshot, OPD computes:

$$p_{i,t}(v) = \pi_{\theta_{\text{old}}}\left(v \mid x_i, \hat{a}_i, y_{i,<t}\right), q_{i,t}(v) = \pi_{\theta_{\text{old}}}\left(v \mid x_i, a_i^*, y_{i,<t}\right), s_{i,t}^{\text{OPD}} = q_{i,t} - p_{i,t}. \quad (1)$$

The difference $s_{i,t}^{\text{OPD}}$ is a signed token-level advantage. A positive gap means that the verified evaluation increases support for token v, while a negative gap means that it decreases that support. Under the GRPO or policy-gradient objective, positive advantages increase the token's log-probability and negative advantages decrease it (Shao et al., 2024; Zhong et al., 2026; Lu et al., 2026b). OPD therefore reinforces planning decisions supported by the verified evaluation and suppresses contradictory decisions without replacing the student's prefix or exposing verified labels at deployment.

Noise-aware asymmetric OPD. Recent work uses token entropy as a proxy for decision uncertainty and shows that probability-based distillation signals become less reliable under prefix mismatch and policy noise (Li et al., 2026; Fu et al., 2026b). In our data, an incorrect evaluation prefix often produces a flatter and less concentrated distribution over subsequent planning tokens.

In low-entropy regions, the model concentrates probability mass on a specific decision region. These tokens may overfit an erroneous region, but when the OPD advantage agrees with the task-level RL reward, both objectives provide consistent evidence. The signed OPD signal then correctly increases or decreases the probability of the corresponding planning token, so low-entropy signals are retained. In high-entropy regions, the model is uncertain and the flatter distribution makes probability gaps more sensitive to prefix noise. The OPD advantage may then conflict with the task-level RL reward, destabilizing training. We therefore attenuate negative OPD advantages in high-entropy regions while preserving positive evidence:

$$h_{i,t} = -\sum_{v} p_{i,t}(v) \log p_{i,t}(v), g_{i,t} = \sigma((\kappa - h_{i,t})/\beta), \delta_{i,t}^{\mathrm{OPD}} = [s_{i,t}^{\mathrm{OPD}}]_{+} + g_{i,t}[s_{i,t}^{\mathrm{OPD}}]_{-}. \quad (2)$$

We call the resulting method **Noise-Aware Entropy-Gated Policy Optimization (NEPO)**. The complete response receives the task-level VQA or RL reward, while each planning token additionally receives $\lambda_{\mathrm{OPD}}\delta_{i,t}^{\mathrm{OPD}}$ under the standard GRPO objective. Only Embodied-Corrector is updated. At deployment, the verified-evaluation branch is removed, and one inference generates the complete Judge–Cause–Sub–Fine response.

3 EXPERIMENTS

We organize six experiments according to three evaluation levels. First, brain-level experiments evaluate execution understanding and corrective planning on our EmbodiedDiag-Bench, followed by transfer to the external EB-ALFRED benchmark. Second, Harness-level experiments use two mutually exclusive execution settings: a code workflow with the VLA function disabled, and a VLA-only workflow in which all other robot functions are disabled. Third, we compare the optimization dynamics of different VQA post-training objectives. A sixth experiment, reported in Appendix 3.4, evaluates the frozen corrective brain as a subtask reward model for VLA training.

3.1 BRAIN-LEVEL EVALUATION

Corrective reasoning on EmbodiedDiag-Bench. We first remove the code and action executors and directly test whether the brain can produce a coherent correction from a completed physical execution. EmbodiedDiag-Bench contains held-out navigation and manipulation failures and evaluates four fields. Judge determines whether the current subtask was completed, Cause identifies the failed entity, relation, and outcome, Sub decides whether to advance or complete the current subtask again, and Fine specifies the required actions, grounded arguments, ordering, and intermediate checks.

Table 1: Brain-level corrective reasoning on EmbodiedDiag-Bench (%). Judge and Cause measure whether the brain understands the realized execution, while Sub and Fine measure whether that understanding produces the correct continuation.

Model	Navigation				Operation			
	Sub	Fine	Judge	Cause	Sub	Fine	Judge	Cause
GPT-5.5	72.7	39.4	52.8	39.8	65.4	41.1	57.7	43.2
RoboBrain2.5-4B	26.1	21.2	31.5	12.2	29.7	24.1	33.5	12.3
RynnBrain-8B	23.4	19.4	23.6	13.6	31.6	25.8	31.9	18.7
InternVL3-8B	21.8	17.7	48.3	26.2	32.1	26.8	40.1	17.9
MiMo-Embodied-7B	31.6	25.3	30.1	16.9	35.6	27.6	22.2	19.3
Qwen2.5-VL-7B	23.2	19.3	52.5	23.8	28.8	26.0	33.7	13.5
Qwen3-VL-32B	48.7	39.8	49.5	29.2	42.9	34.3	54.6	33.2
RynnBrain-SFT	68.6	66.7	75.9	53.4	62.5	61.6	80.7	51.4
RynnBrain-OPD	70.0	68.0	79.0	57.0	**65.4**	61.0	80.0	54.0
RynnBrain-GRPO	**73.0**	69.2	74.2	51.5	63.6	62.1	76.3	49.6
RynnBrain-NEPO	72.6	**71.6**	**79.6**	**58.2**	64.9	**63.7**	**81.2**	**54.6**
- w/o Ent.-Gate	69.9	68.5	73.1	50.8	63.1	61.4	75.0	48.7

Table 1 shows that general VLMs can occasionally distinguish success from failure, but their substantially lower Cause and Fine scores indicate that outcome recognition alone does not yield a usable correction. SFT substantially improves consistency among the four fields, while NEPO

provides the most balanced performance across navigation and manipulation. It reaches Judge accuracies of 79.6% and 81.2%, Cause accuracies of 58.2% and 54.6%, and at least 63.7% on all four recovery-planning scores. The model therefore learns a relation between a failure cause and the way the interrupted subtask should be executed.

Transfer to an external planning benchmark. Because EmbodiedDiag-Bench follows the same output structure as the training data, we additionally evaluate frozen checkpoints on EB-ALFRED (Yang et al., 2025a). No explicit failure prompt, code executor, VLA, or corrective Harness is used. All models receive the same instruction, observation, action space, history, and step limit. This experiment tests whether failure-oriented training learns reusable task dependencies rather than only the Judge–Cause–Sub–Fine response format.

Table 2: Planning transfer to EB-ALFRED (%). Improvements on complex, spatial, and long-horizon subsets show that correction training learns reusable task dependencies and action ordering beyond explicit failure prompts.

Model	Base	Common	Complex	Visual	Spatial	Long
GPT-4o	64	54	68	46	52	54
Claude-3.7-Sonnet	68	**68**	70	**68**	62	**70**
Llama-3.2-11B-Vision-Instruct	24	8	16	22	6	6
InternVL3-8B	20	14	14	12	0	2
Ovis2-16B	26	16	24	12	16	4
Qwen3-VL-32B	62	48	46	42	38	46
Gemma-3-12B-IT	32	26	38	22	20	12
RoboBrain2.5-4B	22	16	34	6	16	24
MiMo-Embodied-7B	20	16	24	18	20	18
RynnBrain-8B	30	28	44	26	24	28
RynnBrain-SFT	72	48	62	54	56	52
RynnBrain-OPD	74	50	64	52	58	54
RynnBrain-GRPO	76	54	72	56	60	58
RynnBrain-NEPO	**78**	56	**74**	60	**64**	62
- w/o Ent.-Gate	74	52	68	54	58	56

As shown in Table 2, Embodied-Corrector improves over the base brain on all six subsets. Relative to SFT, the additional gains of NEPO are most pronounced on Complex, Spatial, and Long tasks, which require stronger prerequisite tracking and action ordering. Controlled failure data therefore improves ordinary embodied planning by teaching the model how task progress depends on previously established conditions. The smaller and less structured gains on appearance-dependent cases also indicate that correction training can reason over available visual evidence but cannot replace evidence absent from the observation.

3.2 Harness-Level Evaluation

Brain-level VQA accuracy does not establish that a correction remains useful after workflow generation and physical execution. We therefore evaluate the complete Harness under two mutually exclusive function configurations. The first disables the VLA function and tests correction through analytic robot functions and generated code. The second disables all other robot functions and tests coordination with a frozen VLA. The planning model, memory, execution interface, and budget remain fixed within each comparison.

Code execution without the VLA. We remove f_{VLA} from the function library, so the GPT code generator can invoke only analytic functions for object detection, navigation, pose estimation, arm motion, gripper control, and articulation. Each function returns its arguments, result or exception, and available pre- and post-execution observations, combining an ASPIRE-style execution trace with a CaP-X-style functional robot interface (Lu et al., 2026a; Fu et al., 2026a).

We fix the code model GPT-5.5, function documentation, perception module, tasks, seeds, and execution budget, and replace only the corrective brain. Embodied-Corrector outperforms the untrained brain under all ten conditions. Relative to GPT self-correction, it performs better in

Table 3: Code-Harness with the VLA function disabled. The code model GPT-5.5, analytic robot functions, perception module, and execution budget are fixed; only the corrective brain changes.

Model	OmniBehavior				LIBERO-PRO					
	Rad.-Nav	Rad.-Op	Soda-Nav	Soda-Op	Obj.-P	Obj.-T	Goal-P	Goal-T	Spa.-P	Spa.-T
OpenVLA	4	0	2	0	0	0	0	0	0	0
π_0	8	2	6	0	0	0	0	0	0	0
$\pi_{0.5}$	20	8	18	6	17	1	38	0	20	1
GPT 5.5	76	44	80	60	61	58	72	77	**58**	66
RynnBrain	72	36	72	56	48	46	64	62	45	49
RynnBrain-NEPO	76	40	80	64	**69**	**67**	**82**	**85**	56	**70**

Libero-Pro. Because neither code generation nor the robot functions change, the result shows that a Judge–Cause–Sub–Fine response enables the same workflow model to revise the relevant function calls instead of repeating the original failed program.

Frozen-VLA execution. In the complementary setting, we evaluate HarnessVLA augmented with our Embodied-Corrector, while the execution library contains only f_VLA. Analytic functions for object detection, navigation, pose control, or gripper control are disabled. Embodied-Corrector produces a state-conditioned subtask, fine-grained action text, and execution-error diagnosis and correction; GPT converts these outputs into HarnessVLA code that invokes a frozen π_0 or $\pi_0.5$ through `vla_execute`. The None condition uses the same pipeline without the Corrector. VLA parameters, observations, action interface, and task split are unchanged, so the comparison isolates whether high-level correction and decomposition reduce the execution burden on an existing action model (Zhang et al., 2026b; Nasiriany et al., 2026).

The effect depends strongly on task structure. Atomic-Seen tasks already provide short instructions close to the frozen VLA's training distribution. Reformulating them can introduce an unnecessary language interface error, and atomic success consequently decreases for both policies. Composite-Seen and Composite-Unseen tasks instead require the VLA to map a long compositional instruction directly to actions. Embodied-Corrector converts the instruction into short, state-conditioned subtasks and substantially improves both groups. For $\pi_{0.5}$, Composite-Seen and Composite-Unseen success increases from 7.1% and 1.2% to 12.5% and 7.3%, respectively, producing an overall gain. The result shows that the brain does not replace low-level visuomotor control; it reduces the temporal and compositional complexity that a fixed VLA must solve at once. The atomic degradation identifies brain–VLA language alignment as a remaining system bottleneck.

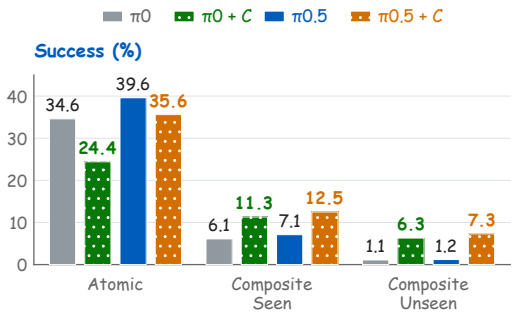

Figure 4: RoboCasa365 success with a frozen VLA as the execution function (%).

3.3 Post-Training Dynamics

We finally examine whether NEPO changes optimization behavior rather than only final benchmark accuracy. GRPO (Shao et al., 2024), GSPO (Zheng et al., 2025), OPD, and NEPO start from the same evaluation-first SFT checkpoint and use the same multimodal VQA data, number of samples, response budget, seed schedule, and update count. A rollout here is one sampled Judge–Cause–Sub–Fine response on a fixed VQA example; it does not involve robot interaction or a multi-turn environment rollout. Figure 5 reports policy loss, answer-level VQA reward, and sampled-token entropy.

Policy loss is non-monotonic for all methods and does not by itself explain their final separation. Vanilla OPD provides dense token-level guidance, but low-quality student prefixes create distribution mismatch between the student and privileged branches. The resulting noisy negative probability gaps



Figure 5: VQA post-training dynamics. NEPO achieves the highest answer-level reward while avoiding the low-entropy regime of vanilla OPD.

gradually move OPD toward a lower-entropy regime. NEPO preserves positive privileged guidance while gating negative signals at uncertain positions. It consequently obtains the highest reward while avoiding the entropy contraction of vanilla OPD. The joint reward–entropy pattern shows that asymmetric gating improves the reliability of OPD supervision.

3.4 Using the Corrective Brain as a Subtask Reward Model

We additionally freeze either the base brain or Embodied-Corrector and use its binary subtask-completion judgment as an auxiliary reward for VLA PPO (Schulman et al., 2017). All conditions share the same policy initialization, terminal environment reward, tasks, interaction budget, and optimization setup. This experiment tests whether the learned judgment can shorten delayed credit assignment rather than merely improve VQA classification.

Table 4: LIBERO with terminal reward and optional frozen-brain subtask reward. Corrective judgments mainly improve Long tasks with delayed dependencies.

Policy	Brain	Spatial	Object	Goal	Long	Avg.
π_0	None	**66**	69	**65**	42	60.5
	RynnBrain-8B	62	66	62	38	57.0
	RynnBrain-NEPO	65	**70**	64	**49**	**62.0**
$\pi_{0.5}$	None	**86**	78	**79**	56	74.8
	RynnBrain-8B	82	75	75	52	71.0
	RynnBrain-NEPO	85	**79**	77	**63**	**76.0**

The trained corrective brain primarily benefits Long tasks, whose intermediate dependencies are difficult to localize from terminal reward alone. On the shorter Spatial, Object, and Goal suites, subtask judgment often duplicates information already contained in the terminal reward and provides limited benefit. Inaccurate judgments from the base brain can instead direct policy updates away from eventual task success. A brain reward is therefore useful when it reliably exposes delayed task structure, neutral when it repeats the environment signal, and harmful when the two disagree.

4 Conclusion and Limitations

We presented Embodied-Corrector, viewing correction not as an auxiliary response to planning failure but as a necessary process for keeping subsequent plans grounded in the physical state. By aligning expected postconditions, observed outcomes, observable failure causes, and recovery decisions, it revises the task state before continuing, converting local deviations into specific recoveries rather than letting them propagate through task dependencies. Its main limitation is that data construction and evaluation remain simulator-based, while observation noise and state uncertainty complicate outcome judgment and recovery verification on physical robots; future work will extend it to real-world deployment via uncertainty-aware postcondition evaluation and safety-constrained recovery.

LLM USAGE STATEMENT

During the preparation of this manuscript, we used OpenAI Codex solely to assist with language editing, grammar correction, and improving the clarity of the writing. OpenAI Codex was not used for research ideation, experimental design, data analysis, or the generation of scientific claims or results. All suggestions were reviewed and revised by the authors, who take full responsibility for the content, accuracy, originality, and integrity of this paper.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*, 2022a.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, et al. Do as I can, not as I say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022b. URL https://arxiv.org/abs/2204.01691.

Anonymous. Unleashing vla potentials in autonomous driving via explicit learning from failures. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026a. URL https://cvpr.thecvf.com.

Anonymous. Embodied failure immunity: Turning robot failures into transferable antigens, 2026b. Zenodo preprint.

Anonymous. Learning from trials and errors: Reflective test-time planning for embodied LLMs, 2026c. URL https://arxiv.org/abs/2602.21198.

Anonymous. From world action models to embodied brains: A roadmap for open-world physical intelligence, 2026d. URL https://arxiv.org/abs/2607.11689.

BAAI RoboBrain Team. Robobrain 2.0: See better. think harder. do smarter., 2026. URL https://superrobobrain.github.io/.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning (CoRL)*, 2023a.

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

Fu Chen, Xin Ding, Bingjia Huang, Xiangyu Li, Mingju Wang, Jiawei He, Kun Li, Wei Sun, Yunxin Liu, Hao Wu, and Ting Cao. Zeva: In-context causal learning for generalizable embodied manipulation. In *arXiv preprint*, 2026. URL https://arxiv.org/abs/2608.30880.

Ronghao Dang, Jiayan Guo, Bohan Hou, Sicong Leng, Kehan Li, Xin Li, Jiangpin Liu, Yunxuan Mao, Zhikai Wang, Yuqian Yuan, Minghao Zhu, Xiao Lin, Yang Bai, Qian Jiang, Yaxi Zhao, Minghua Zeng, Junlong Gao, Yuming Jiang, Jun Cen, Siteng Huang, Liuyi Wang, Wenqiao Zhang, Chengju Liu, Jianfei Yang, Shijian Lu, and Deli Zhao. Rynnbrain: Open embodied foundation models, 2026a. URL https://arxiv.org/abs/2602.14979.

Ronghao Dang, Jiayan Guo, Bohan Hou, Sicong Leng, Kehan Li, Xin Li, Jiangpin Liu, Yunxuan Mao, Zhikai Wang, Yuqian Yuan, et al. RynnBrain: Open embodied foundation models. *arXiv preprint arXiv:2602.14979*, 2026b. URL https://arxiv.org/abs/2602.14979.

Xin Ding, Liang Mi, Mingzhe Huang, Zixuan Wang, Chao Zhang, Zixu Hao, Fu Chen, Xiangyu Li, Yikai Zheng, Yaoyu Guo, Weijun Wang, Kun Li, Hao Wu, Yunxin Liu, and Ting Cao. Zetta ζ: An efficient closed-loop embodied harness for self-evolving physical intelligence. *arXiv preprint arXiv:2608.16590*, 2026. URL https://arxiv.org/abs/2608.16590.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. In *International Conference on Machine Learning (ICML)*, 2023.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi "Jim" Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026a. URL https://arxiv.org/abs/2603.22435.

Yuqian Fu, Haohuan Huang, Kaiwen Jiang, Jiacai Liu, Zhuo Jiang, Yuanheng Zhu, and Dongbin Zhao. Revisiting on-policy distillation: Empirical failure modes and simple fixes. *arXiv preprint arXiv:2603.25562*, 2026b. URL https://arxiv.org/abs/2603.25562.

Liane Galanti, Dhruv Shah, and Tri Dao. Addressing the orchestration gap in generalist robots via physical agency. *arXiv preprint arXiv:2607.21725*, 2026. URL https://arxiv.org/abs/2607.21725.

Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, et al. Octo: An open-source generalist robot policy. In *Robotics: Science and Systems (RSS)*, 2024.

Xintong Hu, Xuhong Huang, Jinyu Zhang, Yutong Yao, Yuchong Sun, Qiuyue Wang, Mingsheng Li, Sicheng Xie, Yitao Liu, Junhao Chen, et al. FineVLA: Fine-grained instruction alignment for steerable vision-language-action policies. *arXiv preprint arXiv:2605.27284*, 2026. URL https://arxiv.org/abs/2605.27284.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning (CoRL)*, 2022a.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022b.

Zitai Huang, Taiyi Su, Jian Zhu, Jianjun Zhang, Chong Ma, Tianbin Liu, Weiyi Lu, Yi Xu, and Hanli Wang. MaskHarness-WAM: Instance-grounded harnessing for long-horizon robot manipulation. *arXiv preprint arXiv:2609.19974*, 2026. URL https://arxiv.org/abs/2609.19974.

Mingcong Lei, Honghao Cai, Yuyuan Yang, Yimou Wu, Jinke Ren, Zezhou Cui, Liangchen Tan, Junkun Hong, Gehan Hu, Shuangyu Zhu, et al. RoboMemory: A brain-inspired multi-memory agentic framework for interactive environmental learning in physical embodied systems. *arXiv preprint arXiv:2508.01415*, 2025. URL https://arxiv.org/abs/2508.01415.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Wensi Ai, Benjamin Martinez, et al. BEHAVIOR-1K: A human-centered, embodied AI benchmark with 1,000 everyday activities and realistic simulation. *arXiv preprint arXiv:2403.09227*, 2024. URL https://arxiv.org/abs/2403.09227.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huanang Gao, Wenkai Yang, Zhiyuan Liu, and Ning Ding. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. *arXiv preprint arXiv:2604.13016*, 2026. URL https://arxiv.org/abs/2604.13016.

Haotian Liang, Mingkang Chen, Yufei Huang, Yuchun Guo, Xiaomeng Zhu, Xiangli Shi, Kaixuan Wang, Yunxuan Mao, Weijie Zhou, Ling Chen, et al. RxBrain: Embodied cognition foundation model with joint language-visual reasoning and imagination. *arXiv preprint arXiv:2607.14187*, 2026. URL https://arxiv.org/abs/2607.14187.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023a.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, et al. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation*, 2023b.

Huihan Liu, Shivin Dass, Roberto Martín-Martín, and Yuke Zhu. Model-based runtime monitoring with interactive imitation learning. *arXiv preprint arXiv:2310.17552*, 2023. URL https://arxiv.org/abs/2310.17552.

Xuqi Liu, Minghe Gao, Juncheng Li, and Siliang Tang. Neurvla: Unleashing failure-handling capability of vision-language-action models via neural-symbolic reasoning. In *International Conference on Machine Learning (ICML)*, 2026a. URL https://icml.cc/virtual/2026/poster/63650.

Yang Liu, Weixing Chen, Xinshuai Song, Tao Pu, Siwen Mo, Yongjie Bai, Zihao Chen, Qianran Sun, Liruo Zhong, Ying Shen, and Liang Lin. PhyAgentOS: A self-evolving operating system for embodied agents with decoupled cognitive planning and physical execution. *arXiv preprint arXiv:2607.16636*, 2026b. URL https://arxiv.org/abs/2607.16636.

Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, Mosharaf Chowdhury, Yuke Zhu, Linxi "Jim" Fan, and Guanzhi Wang. ASPIRE: Agentic /skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026a. URL https://arxiv.org/abs/2607.00272.

Zhengxi Lu, Zhiyuan Yao, Zhuowen Han, Zi-Han Wang, Jinyang Wu, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Self-distilled agentic reinforcement learning. *arXiv preprint arXiv:2605.15155*, 2026b.

Xiaoyu Ma, Lianyu Hu, Wenbing Tang, Zixuan Hu, Zeqin Liao, Zhizhen Wu, and Yang Liu. BrainMem: Brain-inspired evolving memory for embodied agent task planning. *arXiv preprint arXiv:2604.16331*, 2026. URL https://arxiv.org/abs/2604.16331.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, et al. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. *arXiv preprint arXiv:2603.04356*, 2026. URL https://arxiv.org/abs/2603.04356.

Abby O'Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, et al. Open x-embodiment: Robotic learning datasets and rt-x models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

OpenGVLab. InternVL3: Advancing the frontier of native multimodal pre-training, 2025. URL https://huggingface.co/OpenGVLab/InternVL3-8B-Instruct.

Yi Pan, Miao Pan, Qi Lu, Jiaming Huang, Man Zhang, Siteng Huang, Xin Li, Jie Zhang, Yongliang Shen, Xuhong Zhang, and Wenqi Zhang. VLA-Corrector: Lightweight detect-and-correct inference for adaptive action horizon. *arXiv preprint arXiv:2607.01804*, 2026. URL https://arxiv.org/abs/2607.01804.

Physical Intelligence Team. Survey of π_0, π_0-FAST, and $\pi_{0.5}$: Vision-language-action models in the physical AI framework. In *IEEE International Conference on Big Data and Smart Computing (BigComp)*, Jeju, Korea, 2026. URL https://ieeexplore.ieee.org/document/11388479.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *Transactions on Machine Learning Research (TMLR)*, 2022.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023b.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Kaustubh Sridhar, Souradeep Dutta, Dinesh Jayaraman, and Insup Lee. Ricl: Adding in-context adaptability to pre-trained vision-language-action models. In *Conference on Robot Learning (CoRL)*, volume 305 of *Proceedings of Machine Learning Research*, pp. 5022–5038, 2025. URL https://proceedings.mlr.press/v305/sridhar25a.html.

VeBrain Team. Visual embodied brain-1.5: Enhanced perception, spatial reasoning and robot control in spaces. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026. IEEE Xplore.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Qi Wang, Tianyi Wang, Chengyang Li, Shikun Ban, Yurun Chen, Yizhong Ge, Jason Qin, Chengtai Li, and Wentao Zhu. Towards the harness of embodied agents. *arXiv preprint arXiv:2608.11246*, 2026. URL https://arxiv.org/abs/2608.11246.

Xiaomi MiMo Team. MiMo-Embodied: X-embodied foundation model technical report, 2026. URL https://huggingface.co/XiaomiMiMo/MiMo-Embodied-7B.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, et al. EmbodiedBench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. *arXiv preprint arXiv:2502.09560*, 2025a. URL https://arxiv.org/abs/2502.09560.

Rui Yang, Hanyang Chen, Junyu Zhang, Yuchen Zhao, Yifan Zhang, Yixin Chen, et al. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. In *International Conference on Machine Learning (ICML)*, 2025b.

Shuai Yang, Hao Li, Bin Wang, Yilun Chen, Yang Tian, Tai Wang, Hanqing Wang, Feng Zhao, Yiyi Liao, and Jiangmiao Pang. Vision-language-action instruction tuning: From understanding to manipulation. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tsxwloasw5.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, et al. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Chenduo Ying et al. Robofailring: Timely failure detection and causal reasoning in robotic manipulation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026. URL https://aclanthology.org/2026.acl-long.602/.

Wenqi Zhang, Mengna Wang, Gangao Liu, Xu Huixin, Yiwei Jiang, Yongliang Shen, et al. Embodied-Reasoner: Synergizing visual search, reasoning, and action for embodied interactive tasks. *arXiv preprint arXiv:2503.21696*, 2025. URL https://arxiv.org/abs/2503.21696.

Wenqi Zhang, Mengna Wang, Gangao Liu, Huixin Xu, Yiwei Jiang, Yongliang Shen, Guiyang Hou, Zhe Zheng, Hang Zhang, Xin Li, Jiajun Liu, Weiming Lu, Peng Li, and Yueting Zhuang. Embodied-reasoner: Synergizing visual search, reasoning, and action for embodied interactive tasks. In *International Conference on Learning Representations (ICLR)*, 2026a. URL https://iclr.cc/virtual/2026/10012504.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, et al. Harness VLA: Steering frozen VLAs into reliable manipulation primitives via memory-guided agents. *arXiv preprint arXiv:2607.08448*, 2026b. URL https://arxiv.org/abs/2607.08448.

Yue Zhang, Rui Wang, Jiehong Lin, Zhongrui Wang, and Xiaojuan Qi. Retrieval-vla: Training-free in-context adaptation for vision-language-action models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Findings*, pp. 1358–1367, 2026c. URL https://openaccess.thecvf.com/content/CVPR2026F/html/Zhang_Retrieval-VLA_Training-Free_In-Context_Adaptation_for_Vision-Language-Action_Models_CVPRF_2026_paper.html.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025. URL https://arxiv.org/abs/2507.18071.

Zhide Zhong, Haodong Yan, Junfeng Li, Junjie He, Tianran Zhang, and Haoang Li. VLA-OPD: Bridging offline SFT and online RL for vision-language-action models via on-policy distillation. *arXiv preprint arXiv:2603.26666*, 2026. URL https://arxiv.org/abs/2603.26666.

A Related Work

Embodied agent architectures. Language-model robot systems commonly separate semantic planning from physical execution (Wang et al., 2026; Galanti et al., 2026; Liu et al., 2026b). SayCan grounds skill selection with learned affordances, Inner Monologue returns environment feedback to the planning context, and ReAct alternates reasoning with tool use (Ahn et al., 2022b; Huang et al., 2022b; Yao et al., 2023). Code as Policies turns language plans into executable programs, while CaP-X evaluates code agents through perception, geometry, and manipulation interfaces (Liang et al., 2023b; Fu et al., 2026a). Harness VLA routes plans to frozen VLA skills, and FineVLA and InstructVLA show that language guidance must retain execution-critical attributes (Zhang et al., 2026b; Hu et al., 2026; Yang et al., 2026). These systems motivate our fixed execution backends, but our central question is upstream: how failure evidence should change the brain plan before the backend attempts the next recovery.

Embodied execution correction. Correction begins with feedback grounded in what the robot actually did (Huang et al., 2026). Inner Monologue feeds success, scene, and human feedback back into high-level planning, while model-based runtime monitoring detects anomalous or risky execution states (Huang et al., 2022b; Liu et al., 2023). At the controller level, VLA-Corrector compares predicted and observed visual dynamics, truncates stale action chunks, and triggers local replanning (Pan et al., 2026). Language-agent methods such as Reflexion and Self-Refine revise a prior answer from textual feedback (Shinn et al., 2023b; Madaan et al., 2023), but their feedback need not identify a physically observable cause or verify that a repair changes the environment. ASPIRE comes closest to that closed loop: its execution engine exposes fine-grained multimodal traces for failure diagnosis, repair synthesis, and validation (Lu et al., 2026a). Embodied-Reasoner, RynnBrain, and RxBrain additionally provide multimodal task reasoning and execution evaluation (Zhang et al., 2025; Dang et al., 2026b; Liang et al., 2026). Our focus is the missing task-level connection among these capabilities: after each completed subtask, identify the observable cause, revise the remaining dependency plan from the current state, and verify that the same fixed executor realizes the correction.

Memory for embodied agents. Voyager stores reusable skills; BrainMem separates working, episodic, and semantic memory; RoboMemory retrieves procedural experience; and Harness VLA retains execution traces (Wang et al., 2023; Ma et al., 2026; Lei et al., 2025; Zhang et al., 2026b). These systems show that memory supports long-horizon state tracking, similar-task planning, and skill reuse, but retrieval alone does not show that a repeated physical error was prevented. Our architecture therefore separates volatile working state from environment-validated correction episodes and only consolidates a semantic rule after multiple compatible successes. This write policy treats memory as evidence-backed context for correction rather than as authority over the current observation.

B Implementation Details

B.1 Data Composition

Training uses two sources: our controlled correction data and the Embodied-Reasoner corpus. The controlled data start from 300 RoboCasa365 tasks, containing 6,210 episodes and 2,078,151 raw rounds, and 50 OmniBehavior tasks, containing 328 episodes and 358,608 raw rounds (Nasiriany et al., 2026; Li et al., 2024). We inject controlled perturbations at subtask boundaries and construct Judge, Cause, Sub, and Fine supervision from the resulting executions. After filtering for effective perturbations, complete observations, observable failures, and duplicate trajectories, the corpus contains 1,583,014 valid rounds from 148,415 rollouts. Source trajectories are split before perturbation branches are generated, preventing examples with the same successful prefix from appearing in both the training set and EmbodiedDiag-Bench.

Evaluation-first SFT mixes the controlled correction data with general embodied-planning data from Embodied-Reasoner. The controlled data teach execution judgment, failure-cause analysis, and corrective planning, whereas Embodied-Reasoner preserves general task decomposition, object-relation reasoning, and action-dependency knowledge (Zhang et al., 2025).

The second stage also operates on fixed multimodal VQA examples and contains no robot interaction or environment rollout. Its data consist of 28,871 correction responses sampled from the student distribution and 3,003 Embodied-Reasoner planning examples, producing 31,874 records in total. All post-training methods share the same data, initialization, prompts, and response budget; they differ only in their reward objective and in whether privileged evaluation and entropy gating are used.

B.2 Unified Harness Evaluation Protocol

Common workflow. All execution experiments use the same "plan–execute functions–correct–revise workflow" loop. A fixed closed-source GPT-5.5 model performs both high-level task planning and workflow generation. It receives the task goal, current observation, within-episode execution history, previous correction, robot-function documentation, and retrieved memory, and produces the next workflow.

Multimodal memory is provided only to GPT-5.5 and is not an input to Embodied-Corrector. Working memory maintains the current observation, completed steps, latest correction, and remaining task. Episodic memory stores environment-validated correction cases, while semantic memory stores cause–repair rules consolidated from repeated successful cases. To keep evaluation trials independent, formal experiments retain only within-episode working memory and disable cross-episode episodic retrieval, memory writing, and semantic-rule updates.

After a robot-function call or VLA action chunk returns, the Harness records pre- and post-execution images, function arguments, return values or exceptions, and observable changes in robot and object state. Following the execution-trace-based correction mechanism of ASPIRE, this multimodal evidence is passed to Embodied-Corrector (Lu et al., 2026a). The corrector receives neither retrieved memory nor hidden simulator state. In a single inference, it produces Judge, Cause, Sub, and Fine: Judge determines whether the current subtask was completed, Cause identifies a failure supported by observable evidence, Sub specifies the next subtask, and Fine describes how that subtask should be executed.

The structured correction is then returned to GPT-5.5. After success, GPT-5.5 preserves completed progress and continues the remaining task. After failure, it revises the subsequent workflow according

to Cause, Sub, and Fine. GPT-5.5 receives only the structured correction record, not the private reasoning of Embodied-Corrector. All quantitative execution experiments are conducted in simulation, and task success is determined only by the public environment predicate of each benchmark.

Code Harness with the VLA function disabled. In the first configuration, f_{VLA} is removed from the function library. GPT-5.5 can compose only analytic robot functions, such as `detect_object`, `navigate_to`, `estimate_pose`, `move_to_pose`, `control_gripper`, and `open_joint`. These functions share a common argument, return, and exception interface and expose multimodal execution traces for correction. This configuration combines a CaP-X-style functional robot interface with ASPIRE-style execution feedback (Fu et al., 2026a; Lu et al., 2026a).

OmniBehavior separately evaluates navigation and manipulation success for the radio and soda-can tasks, giving four metrics: Radio-Navigation, Radio-Operation, Soda-Navigation, and Soda-Operation. Each task–metric condition is evaluated from 25 distinct random initializations. Each reported cell is therefore the mean binary success over 25 complete trials.

The first three rows of Table 3 (OpenVLA, π_0, and $\pi_{0.5}$) are results reported by CaP-X under its original evaluation protocols and were not rerun in our Harness; they are therefore excluded from the 25- and 100-trial settings described above (Fu et al., 2026a).

For the remaining three rows, all compared conditions use the same GPT-5.5 model, robot-function documentation, perception module, task set, initial-state distribution, and execution budget. Only the correction component is changed among GPT-5.5 self-correction, RynnBrain-8B, and Embodied-Corrector. Cross-task memory writing and skill-library updates are disabled, so any change in success measures whether the correction helps the same planner revise subsequent function calls.

HarnessVLA execution with a frozen VLA as the only available robot function. All conditions use HarnessVLA for robot execution. In this configuration, the function library exposed to the GPT code generator contains only `vla_execute`, backed by a frozen π_0 or $\pi_{0.5}$. Analytic functions for object detection, navigation, pose control, gripper control, and articulation are disabled.

At fixed action-chunk boundaries, Embodied-Corrector receives the overall task, the previous stage instruction, and the pre- and post-execution observations. In brain-assisted conditions, it generates a complete Judge–Cause–Sub–Fine record in one inference. This record contains subtask decomposition, fine-grained textual action planning, and execution-error diagnosis and correction. These structured outputs are then passed to GPT, which generates executable HarnessVLA code. Under the frozen-VLA-only configuration, the generated code can invoke only `vla_execute`; in configurations with a larger function library, the same code-generation mechanism can also invoke analytic tools or VLA functions.

The generated code is executed by HarnessVLA. The frozen VLA receives image observations and robot state through the unchanged HarnessVLA interface, together with the language instruction specified by the generated execution call. VLA parameters, observation preprocessing, action space, and task horizon remain unchanged across all conditions.

Evaluation is performed through the same HarnessVLA execution pipeline. It covers 18 Atomic-Seen, 16 Composite-Seen, and 16 Composite-Unseen RoboCasa365 tasks (Nasiriany et al., 2026). Each brain-assisted condition runs 50 independent trials per task, yielding 2,500 episodes for each brain–VLA pair: 900 Atomic-Seen, 800 Composite-Seen, and 800 Composite-Unseen episodes.

C Detailed protocol for the perturbations

Shared intervention protocol. Each example starts from a retained successful trajectory at the state immediately before a subtask boundary. We clone that state and hold the task instruction, successful prefix, intended object and target, and all non-selected execution factors fixed. We then intervene on exactly one of the seven factors below and replay only the affected subtask. An intervention is retained only if the expected postcondition is violated, the consequence is observable in the synchronized public observation or execution record, and the branch remains recoverable. Ineffective interventions, branches with multiple changed factors, and failures that can be identified only from hidden simulator state are discarded. Recovery always begins from the realized failed state rather than from a reset. Because scene geometry, joint ranges, and skill duration differ across tasks, perturbation validity is

defined by the resulting predicate change rather than by one global displacement, angle, or timestep range. This construction makes the factor name a causal intervention label, not a free-form description assigned after a naturally occurring failure.

Navigation. This intervention changes the executed navigation goal, the desired terminal heading, or both, while leaving the semantic destination and manipulation request unchanged. It is applied at a navigation boundary before the affected navigation skill is replayed. The resulting stop can lie outside the interaction region or face the workspace from an incorrect approach direction, so the target-region or pose postcondition fails even though locomotion itself may finish normally. A matched correction estimates a new reachable goal and heading from the current observation, navigates there, verifies that the interaction region is in range, and then resumes the blocked manipulation. This factor tests whether the brain can distinguish successful motion from task-relevant arrival and repair a geometric navigation error instead of repeating the same command.

Reachability. Reachability keeps the intended destination fixed but makes the original waypoint or terminal stop pose unusable. The intervention either blocks that pose or invalidates it under the planner's collision and kinematic feasibility checks. Unlike Navigation, which perturbs where or how the agent is asked to arrive, Reachability perturbs whether the nominal route or stop configuration can be executed. The observed consequence is a no-path result, an unreachable target, or a stop from which the next end-effector goal cannot be reached. Recovery resamples a collision-free intermediate waypoint or stop pose, verifies planning feasibility, and returns to the original subtask. The diagnostic requirement is to propose an alternative feasible route or base configuration rather than misclassifying the failure as object absence or a grasp error.

Grasp pose. This intervention changes the end-effector approach direction, contact height, or grasp orientation for the specified object while preserving object identity and the requested downstream task. The perturbed grasp can miss the object, contact an unstable region, or initially contact and then slip. Offline construction uses pose, contact, and attachment signals to confirm the label, while the deployed brain must infer the failure from public images and the execution record. The correction re-localizes the object, recomputes a reachable pre-grasp and grasp pose, approaches without collision, closes the gripper, and verifies attachment before lifting or transporting. This factor evaluates spatial diagnosis at the object–gripper interface and whether the recovery restores a stable grasp rather than merely issuing another close command.

Timing. Timing changes when the gripper open or close command occurs relative to approach, contact, lift, transport, or release. The geometric target and grasp pose remain fixed, isolating temporal coordination from spatial accuracy. An early close can prevent the gripper from enclosing the object; a late close can begin after contact has been lost; an early open can release during transport; and a late open can leave placement incomplete. The common observable outcomes are an unsecured object or a drop. Recovery first restores the relevant pre-contact or pre-release relation, then issues the gripper command at the correct phase and verifies attachment or release before continuing. This factor tests whether the brain attributes failure to action order and synchronization rather than unnecessarily changing the object, goal, or grasp geometry.

Container. The Container intervention changes the articulation state of a receptacle such as a drawer, cabinet, or door by leaving it only partially open or partially closed at the subtask boundary. The container identity and requested content operation remain unchanged, but the opening, clearance, or required joint-state predicate is absent. The next reach, placement, retrieval, or closure can therefore be infeasible even if its own command is well formed. A matched correction re-localizes the articulated part, completes the required open or close motion past the task-defined postcondition, verifies the resulting space or state, and resumes the blocked content operation. This factor tests whether the brain recognizes articulation as a physical precondition rather than repeatedly repairing the downstream grasp or placement.

Prerequisite. This intervention removes one dependency from the executed task prefix while keeping the blocked subtask request unchanged. Typical dependencies include reaching the workspace before manipulation, opening a container before insertion or retrieval, holding an object before transport, or clearing a destination before placement. The requested subtask may be syntactically

valid but is infeasible in the realized state because its first required predicate is false. Recovery identifies the earliest unmet dependency, executes and verifies it, and then returns to the original blocked subtask without discarding the successful prefix. This factor directly measures causal task-graph reasoning: the correct response must repair an earlier condition, not paraphrase or locally retry the final command.

Target. Target intervention changes the grounded same-class instance or the executable reference while preserving the task instruction and skill type. For example, a command intended for one bowl can be bound to another visible bowl, so a technically successful grasp or placement still changes the wrong entity. The failure is accepted only when instance identity and the task relation disagree and the mismatch is observable in the retained evidence. Recovery resolves the instruction against the current scene, re-grounds the specified instance and destination, undoes or safely terminates the wrong-object branch when necessary, and replans the blocked operation. This factor tests reference resolution and instance-level grounding, separating semantic target errors from failures of navigation, reachability, or control.

D CORRECTIVE HARNESS

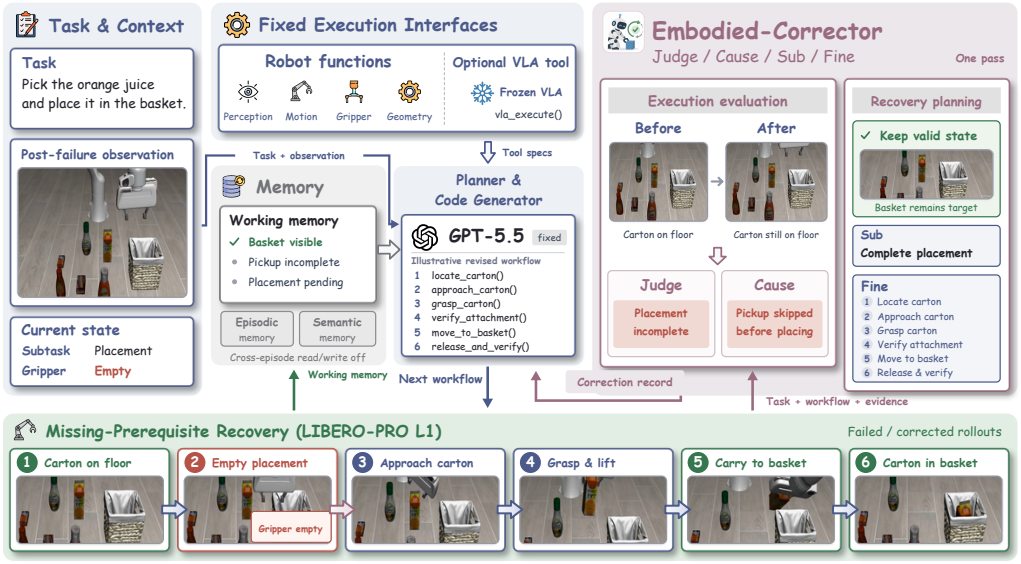

Figure 6: Architecture and closed-loop workflow of the Embodied-Corrector Harness. Separating execution evaluation, recovery planning, fixed code/VLA realization, and environment-validated memory turns failure evidence into executable correction while preserving prior progress.

Problem formulation. Given a long-horizon task goal g, at execution round t, a fixed closed-source GPT model receives the current observation o_t, execution history H_t, retrieved memory $\mathcal{R}(M_t)$, the previous correction record e_{t-1}, and the robot-function documentation $\mathcal{D}_{\mathcal{F}}$. It outputs an executable workflow w_t. After the workflow is executed, the Harness returns the new observation o_{t+1} and execution trace τ_t. Embodied-Corrector then produces a structured correction record from the resulting execution evidence:

$$w_t = \mathrm{GPT}(g, o_t, H_t, \mathcal{R}(M_t), e_{t-1}, \mathcal{D}_{\mathcal{F}}), e_t = (d_t, c_t, r_t) = \mathcal{C}_\theta(g, w_t, o_t, o_{t+1}, \tau_t). \quad (3)$$

Here, w_t denotes workflow code composed of robot-function calls, and $\mathcal{D}_{\mathcal{F}}$ describes callable functions for object detection, navigation, pose control, gripper control, articulation, and a frozen VLA. The outputs d_t, c_t, and r_t denote the subtask completion judgment, the failure cause supported by execution evidence, and the recovery proposal from the current state, respectively. Retrieved memory is provided only to the GPT model, whereas Embodied-Corrector reasons only from the task, the executed workflow, the pre- and post-execution observations, and the returned trace. The GPT

model then conditions on e_t in the next round to continue or revise the workflow while preserving valid prior progress.

Workflow. As illustrated in Figure 6, the closed-source planning model first retrieves task-relevant context from the multimodal memory shown on the left. Working memory maintains the current task state, the latest execution outcome, and the unfinished subtask. Episodic memory stores environment-validated successful corrections, such as a case in which a skipped grasp must be inserted before transport. Semantic memory consolidates reusable task rules, such as requiring object attachment before transport and verifying that attachment before proceeding. These memories are provided only to the closed-source planning and workflow-code model, not to Embodied-Corrector. When retrieved experience conflicts with the current observation, the current physical evidence takes precedence.

The closed-source planning model combines the task goal, current state, and retrieved memory to directly generate a workflow over robot functions. Following the function-based robot interface of CaP-X, the function set $\mathcal{F}$ may include `detect_object`, `navigate_to`, `approach`, `close_gripper`, `lift`, `open_joint`, and `verify_attachment` (Fu et al., 2026a). Following Harness VLA, a frozen VLA is exposed through the same function library as `vla_execute(instruction, observation, robot_state)`, which realizes local contact-rich skills such as grasping, placing, and articulation (Zhang et al., 2026b). The VLA is therefore not a separate execution system outside the workflow. It is one robot function, alongside navigation, pose control, gripper control, and other analytic functions.

After a robot function or VLA function is executed, the fixed execution Harness returns the pre- and post-execution observations, function arguments, return value or exception, and observable changes in robot and object state. Following the execution-trace-based correction mechanism of ASPIRE, this multimodal evidence is passed to Embodied-Corrector (Lu et al., 2026a). The corrector neither retrieves memory nor directly generates executable workflow code. Instead, it determines whether the requested subtask was actually completed, identifies a failure cause supported by the observations, and proposes the task condition or recovery steps that must be restored. In the LIBERO-PRO orange-juice example in the figure, the failed program attempts placement with an empty gripper while the carton remains on the floor. The corrector therefore marks placement as incomplete, identifies the omitted pickup prerequisite, and proposes locating and grasping the carton, verifying attachment, and placing it in the basket. The lower strip shows frames from the failed rollout followed by frames from the regenerated rollout, rather than one uninterrupted trajectory.

The correction record is then returned to the same closed-source planning model. After success, the planner preserves completed progress and continues the remaining workflow. After failure, it combines the recovery proposal with retrieved memory to regenerate the required function calls—for example, target localization, approach, gripper closure, lifting, and attachment verification before resuming placement. The new physical outcome is evaluated again, forming a loop of "memory retrieval–workflow generation–function execution–execution correction–workflow revision." Working memory is updated after every round. A correction enters episodic memory only after the environment confirms that it restored task progress, and repeated compatible successes may later be consolidated into semantic rules. During formal evaluation, the closed-source planner, robot functions, frozen VLA, perception interface, and execution budget are held fixed, so the experiment measures whether Embodied-Corrector can restore task progress by changing the subsequent workflow.

E Public Robot Tool Interfaces for Closed-Loop Correction

E.1 Overview

At the beginning of each subtask, the code-generation model receives a complete list of tool interfaces, including function signatures, parameter meanings, return values, coordinate frame conventions, and major failure conditions. The model is not allowed to send undefined robot control commands directly; instead, it generates Python programs composed solely of these public tools. This design decouples high-level reasoning from low-level execution: the "brain" is responsible for determining whether a subtask is completed, analyzing failure reasons, and generating fine-grained recovery plans; the code model translates recovery plans into calls for perception, geometric computation, grasp planning, and motion control.

The tool interfaces also serve as the adaptation boundary between simulation and real hardware. The simulation and real-robot backends may use different camera drivers, segmentation services, inverse kinematics solvers, and arm controllers, but they must preserve identical tool names, parameter semantics, coordinate frames, and return structures. Currently, the LIBERO/ASPIRE backend provides 24 public functions, among which point_prompt_molmo is registered only when the Molmo service is available. The code-generation model may define auxiliary functions local to the task, but it cannot bypass these interfaces to access private simulator states.

Environment success determination is not part of the tool interfaces. The generated program can only read public observations and robot states; it cannot access rewards, task-completion flags, ground-truth object poses, or internal contact states of the simulator. After the program terminates, an independent environment evaluator computes success or failure based on task predicates and returns the result to the Harness. Therefore, the model's own completion declarations do not directly affect the experimental results.

E.2 Public Tool Interface Summary

The following public tools are available to the code-generation model. The interface covers visual perception, 3D geometry, grasp planning, and robot control. All state-changing operations must be executed through the motion-control tools, while environment rewards and task-completion predicates remain hidden from the generated program.

Observation

- **get_observation()** – Returns a multimodal observation dictionary containing RGB, depth, intrinsics, and camera-to-world transforms from the main and wrist cameras, along with robot joint angles, end-effector pose, and gripper state. This is the sole entry point for obtaining the current environment state.

Vision

- **segment_sam3_text_prompt(rgb, text_prompt)** – Returns a list of mask candidates (each with binary mask, bounding box, and confidence) for the text-described instance; returns an empty list if no target is detected.
- **segment_sam3_point_prompt(rgb, point_coords)** – Segments the instance at the given pixel coordinate, useful when text-based segmentation is unstable or disambiguation is needed.
- **point_prompt_molmo(image, text_prompt)** – Returns the pixel coordinates of a text-described target using a vision-language model; registered only when the Molmo service is available.

Grasp Planning

- **plan_grasp(depth, intrinsics, segmentation)** – Generates grasp candidates (transform matrix and confidence) from a depth image and target mask using Contact-GraspNet. Returned poses are in the camera frame and must be transformed to the world frame; reports failure if no candidate.
- **plan_grasp_from_point_clouds(pc_full, pc_segment)** – Generates grasp candidates from the full scene point cloud and target object point cloud, suitable for multi-view fusion.
- **select_top_down_grasp(grasps, scores, cam_to_world, vertical_threshold=0.8)** – Transforms candidates to the world frame, filters for near-vertical grasps, and returns the highest-scoring valid candidate.

Geometry

- **depth_to_point_cloud(depth, intrinsics)** – Back-projects the entire depth map into a point cloud in the camera frame using the pinhole model.

- **mask_to_world_points(mask, depth, intrinsics, extrinsics)** – Back-projects valid depth pixels within the mask to 3D and transforms them to the world frame.
- **pixel_to_world_point(u, v, z, intrinsics, extrinsics)** – Converts a single pixel and its depth to a 3D world point, useful for turning vision model pointing into robot targets.
- **get_oriented_bounding_box_from_3d_points(points)** – Computes the oriented bounding box (center, extent, rotation matrix) of the target point cloud.
- **interpolate_segment(p1, p2, step=0.03)** – Generates evenly spaced waypoints between two endpoints, used for staged approach, lifting, transport, or pushing.
- **normalize_vector(v)** – Normalises a direction vector; near-zero vectors are left unchanged to avoid numerical issues.

Point Cloud Processing

- **subsample_point_cloud(pc, max_points=10000)** – Downsamples the point cloud to reduce time and memory overhead.
- **filter_noise(points, colors=None)** – Removes outliers using density-based clustering; preserves colour correspondence if provided.

Transformations

- **transform_points(points, transform_matrix)** – Applies a 4×4 homogeneous transform to a point set or organised point cloud.
- **rotation_matrix_to_quaternion(R)** – Converts a 3×3 rotation matrix to a unit wxyz quaternion.
- **decompose_transform(T)** – Decomposes a 4×4 homogeneous grasp transform into position and wxyz quaternion for control interfaces.

Motion Planning and Control

- **solve_ik(position, quaternion_wxyz)** – Solves inverse kinematics for the Franka arm (7-DOF). Falls back to predefined poses if the target is unreachable; reports error if all fail.
- **move_to_joints(joints)** – Blocking move to the specified joint configuration; returns after motion completes.
- **goto_pose(position, quaternion_wxyz, z_approach=0.0)** – Wraps IK and joint motion. When `z_approach` is set, moves to a point above the target then descends.

State Recovery

- **goto_home_joint_position()** – Returns the arm to a predefined home configuration. Should not be called while carrying an object.

Gripper Control

- **open_gripper()** – Fully opens the gripper.
- **close_gripper()** – Fully closes the gripper. Closing alone does not imply successful grasping; subsequent observations must verify stable lifting.

E.3 Typical Tool Calling Workflow

For a pick-and-place subtask, the generated program first calls get_observation to acquire images and camera parameters, then uses text or point prompts to segment the target object. The target mask is converted to a world-frame point cloud via mask_to_world_points, which is used either to estimate the object centre or to generate grasp candidates. Subsequently, the program calls select_top_down_grasp, decompose_transform, and solve_ik to obtain an executable pose, then executes the grasp through open_gripper, goto_pose, close_gripper, and a lifting motion. After placement, the program calls get_observation again to acquire evidence of the outcome. Only when the external environment evaluator confirms that the task predicate holds does the Harness mark the subtask as successful.

E.4 Tool Call Logging and Error Correction

Every tool call is written into a structured trajectory that includes at least the call order, function name, parameter summary, execution time, result summary, and corresponding visual keyframes. The correction model receives the task instruction, the failed program, the public observations, and this execution trajectory, and outputs four structured fields: whether the subtask succeeded, the observable failure reason, the next subtask, and a fine-grained recovery plan. The code model then generates a revised program based on this record. In this way, the correction results reported in the paper are grounded in verifiable environmental evidence, rather than in the model's subjective judgment of its own program success.

F CSorrection cases

The following examples show how our **Embodied-Corrector** resolves representative execution failures in LIBERO-PRO and RoboCasa365.

F.1 LIBERO-PRO: correction guides program regeneration

L1, Prerequisite: pick up the orange juice before placement. **Task.** Pick the orange juice and place it in the basket. **Failed execution.** The generated program starts from placement even though the orange-juice carton has not been grasped. The empty gripper moves to the basket and opens, so neither the object state nor the task predicate changes. **Correction.** Embodied-Corrector marks the current subtask as unsuccessful, identifies the omitted pickup as the first causal error, selects "localize and grasp the orange juice" as the next subtask, and specifies grasp, attachment verification, lift, and basket placement in the fine-grained plan. The code generator uses this record to restore the missing dependency instead of refining the already invalid placement motion. **Correct execution.** The regenerated program localizes and grasps the carton, verifies that it moves with the gripper, transports it over the basket opening, and releases it inside. The paired rollout figure contrasts the empty placement attempt with the corrected placement and the environment-verified success state.

L2, Target: resolve the requested bowl before control. **Task.** Pick the black bowl that is *not between* the plate and the ramekin and place it on the plate. **Failed execution.** The generated program admits a dark non-bowl region into its candidate set and plans around that false referent. Although the arm executes plausible motions, acting on the wrong instance cannot satisfy the requested spatial relation. **Correction.** Embodied-Corrector attributes the failure to target selection, retains the placement objective, and asks the code generator to ground the plate, ramekin, and true bowl instances before applying "not between." The fine-grained plan explicitly rejects non-bowl masks and the bowl lying on the plate–ramekin relation. **Correct execution.** The corrected program selects the other black bowl, grasps and lifts it, clears the nearby ramekin, and releases it over the plate center. The final spatial relation is accepted by the environment.

F.2 RoboCasa365: correction revises the instruction to a frozen VLA

In RoboCasa365, control returns to Embodied-Corrector after each VLA action chunk, while the frozen $\pi_{0.5}$ continues to generate low-level actions. Unlike LIBERO-PRO, this evaluation records one continuous rollout rather than a separate failed episode and a restarted successful episode. To make the comparison explicit without inventing a second rollout, each figure below places the observed failed grasp state immediately before correction on the left and the terminal, environment-verified successful state from the same rollout on the right. The complete continuous videos remain the temporal evidence; the paper images are raw observations without labels or annotations.

R1, High and offset grasp: align the avocado. **Task.** Pick the avocado from the sink and place it on the plate located on the counter. **Failed execution.** The rollout reaches over the sink but repeatedly closes without enclosing the avocado. The object remains in the basin, exposing coupled errors in descent and lateral alignment. **Correction.** Embodied-Corrector uses the unsuccessful status, boundary observations, and the history of empty grasp attempts to change the next subtask to side approach and output a directional alignment instruction. As the view changes, the corrector requests small left or right adjustments and only then restores the close-and-lift action. This is an

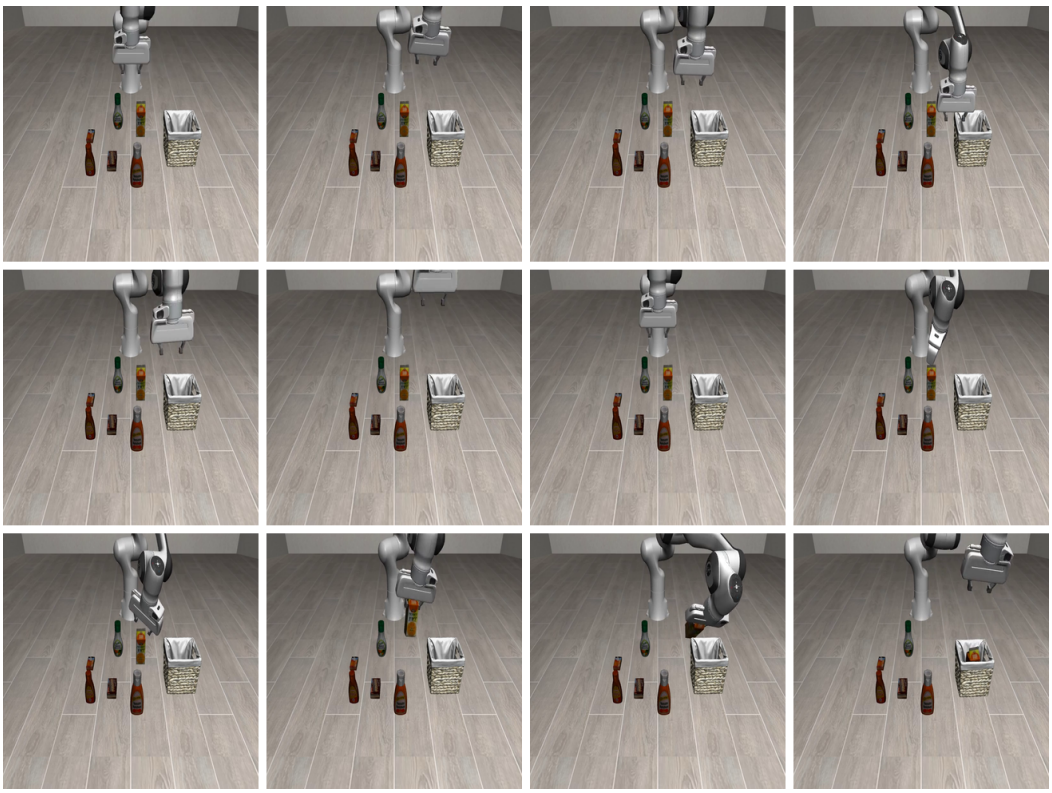

Figure 7: Prerequisite correction in L1. The twelve raw observations show the failed placement attempt followed by the corrected pickup, attachment verification, and placement.

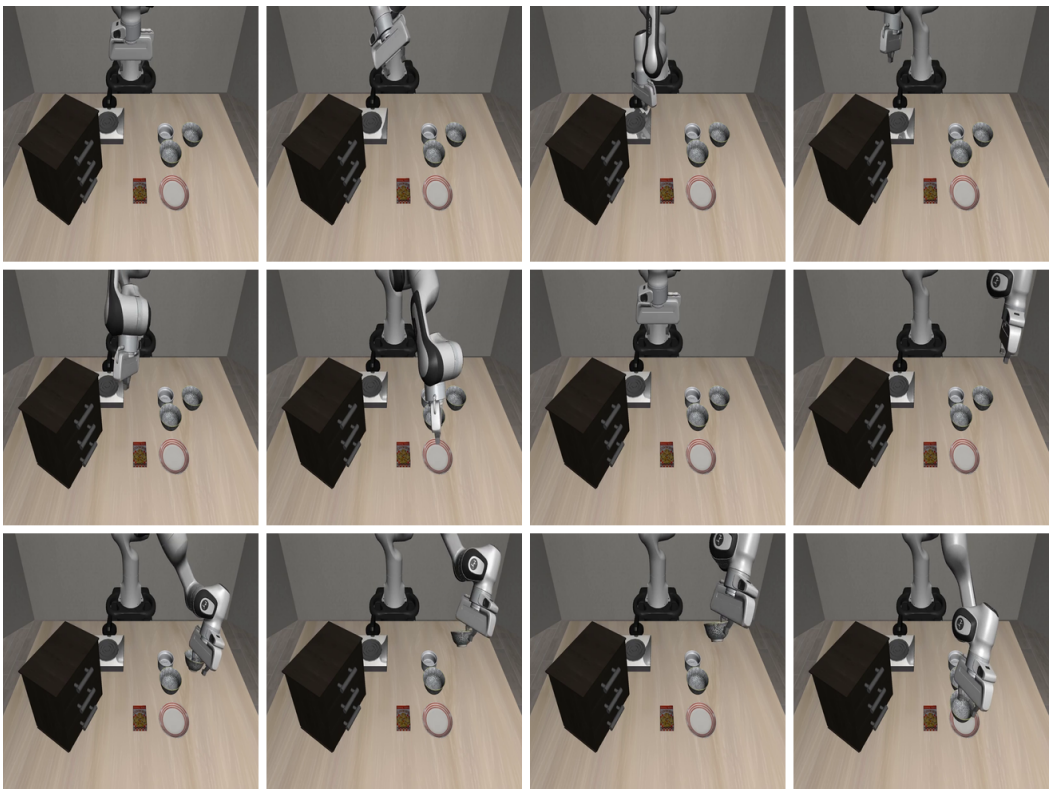

Figure 8: Target correction in L2. The twelve raw observations show the false-target execution followed by re-grounding and verified placement of the requested bowl.

observation-conditioned correction while the VLA remains frozen. **Correct execution.** The corrected instruction brings the gripper into the basin, restores lateral alignment, and leads to the avocado leaving the sink and satisfying the environment's object–plate predicate.

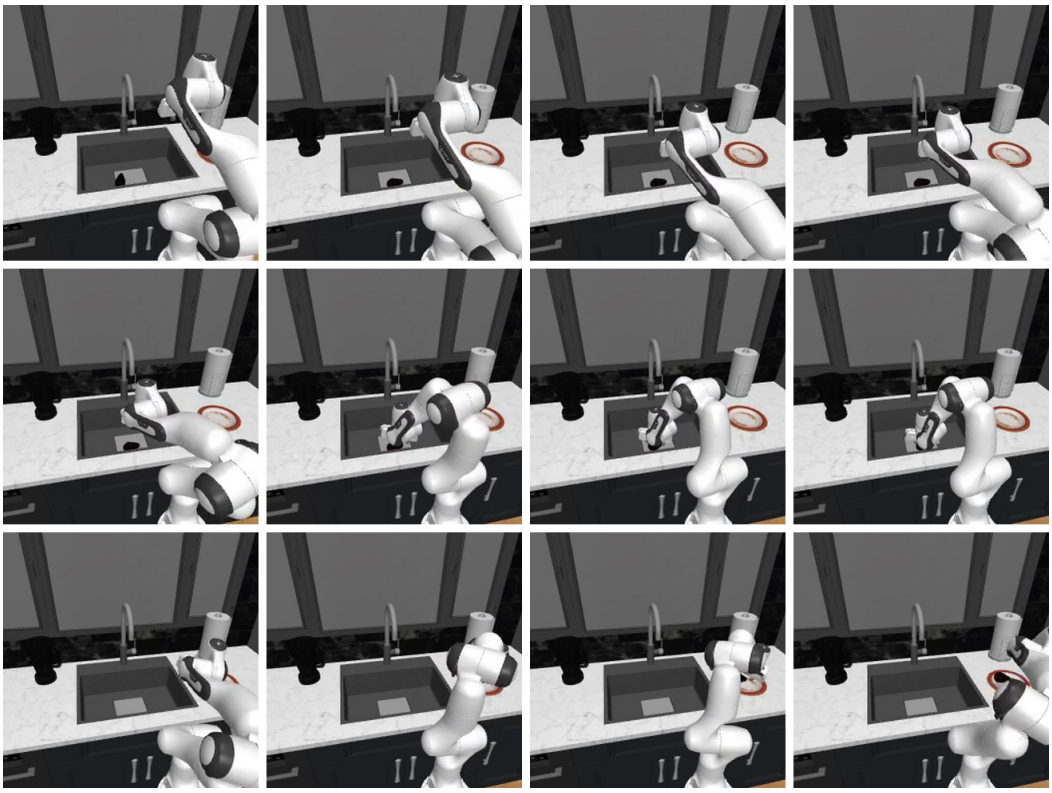

Figure 9: R1 high and offset grasp correction. Twelve raw observations show the gripper remaining too high and laterally misaligned before the directional re-alignment and successful continuation.

R2, Offset and late grasp: reposition before contacting the onion. **Task.** Pick the onion from the sink and place it on the plate located on the counter. **Failed execution.** The first pickup instruction contains directional motion, but the contact timing is wrong: the gripper closes too late, knocks the onion, and leaves it in the sink. Repeating closure without changing the approach does not recover the object. **Correction.** The corrector receives the directional plan, boundary observations, the false success status, and the earlier action history. It keeps the pickup subtask but makes recovery executable by coordinating forward, lateral, and downward repositioning before closure. Later updates continue to adapt the approach as the view changes, rather than repeating an unchanged close command. **Correct execution.** The corrected approach encloses the onion, carries it out of the sink, and satisfies the requested object–plate relation. RoboCasa365 verifies successful termination from its public task predicate.

Across these cases, successful correction depends on replacing the failed execution premise: restore a prerequisite, re-ground the target, or rebuild a stable grasp by changing approach height, lateral alignment, and contact timing. Embodied-Corrector supplies this structured change, while the downstream code generator or frozen VLA executes it and the environment verifies the final result.

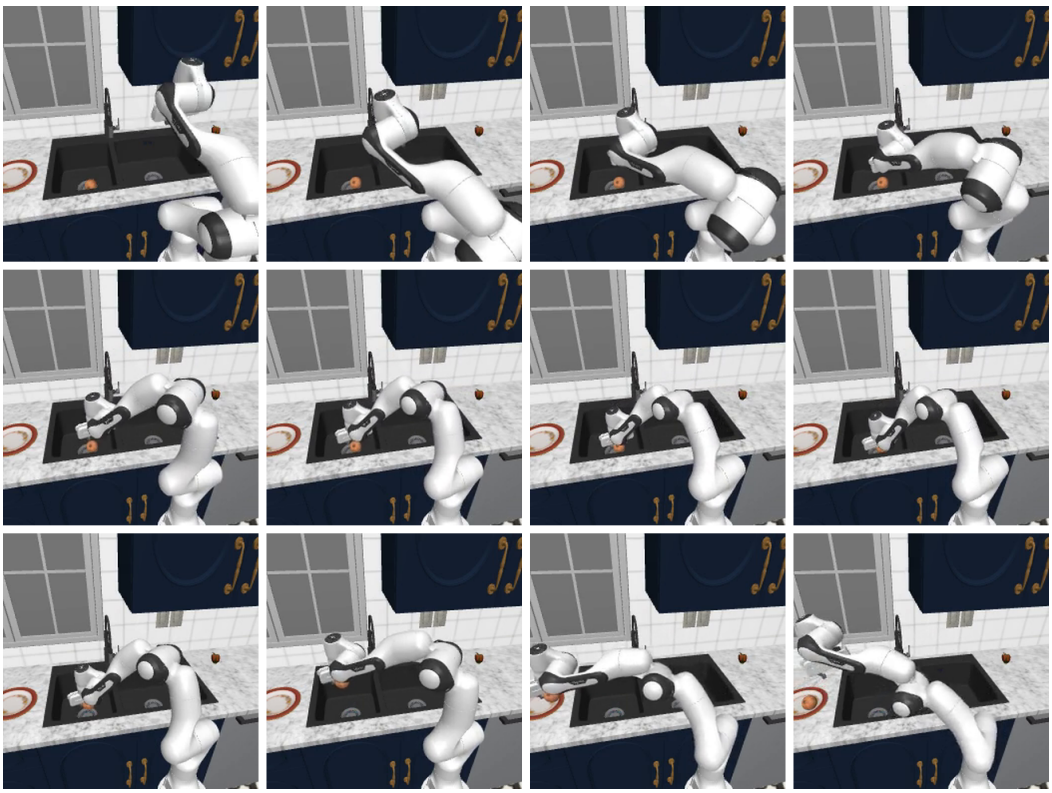

Figure 10: R2 offset and late-contact correction. Twelve raw observations show the late closure and the onion remaining in the sink before coordinated re-approach, closure, and successful continuation.