ENACTOPT: DIAGNOSIS-GUIDED HARNESS EVOLUTION FOR EMBODIED AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Embodied agents built on foundation models rely on an execution harness to organize observations, guide actions, and incorporate execution feedback into subsequent decisions. Failures can arise not only from limited model capabilities, but also from how the harness presents evidence and guides its use. These model-external mechanisms provide an adaptation target without updating model parameters, but task-specific interaction experience can be costly to collect. Task-level outcomes alone do not reveal whether relevant evidence was missing from the model input or was available but inadequately used. Moreover, these two aspects are coupled: effective action guidance may require evidence that the current context omits, while exposing that evidence alone does not ensure an appropriate response. We introduce EnactOpt, which uses limited execution feedback to jointly evolve reusable textual skills and context-construction code while keeping the underlying models and runtime fixed. It diagnoses adjacent decisions by relating action feedback to the actual model input and subsequent action. Structured episode summaries guide joint revisions through prompt-level edit operators, and candidate harnesses undergo compatibility checks and validation-based retention. Across VLABench, ALFWorld, and Embodied Agent Interface (EAI), EnactOpt improves task success by approximately 17–41 percentage points over the respective Naive baselines and outperforms the evaluated GEPA, TextGrad, and Trace2Skill baselines. These gains extend to task conditions not used for harness evolution or candidate selection. The results support a complementary route to embodied adaptation: turning limited execution experience into reusable behavioral improvements by jointly refining how evidence is presented and acted upon.

1 INTRODUCTION

Foundation models provide embodied agents with multimodal reasoning (Driess et al., 2023; Hurst et al., 2024; Bai et al., 2025; Comanici et al., 2025) and sensorimotor control (Kim et al., 2024; Physical Intelligence et al., 2025b; Li et al., 2025; Ye et al., 2026). Effective execution, however, depends on how observations become model inputs, model outputs become actions, and execution feedback informs subsequent decisions (Shi et al., 2025). We call the reusable, model-external guidance and control logic supporting this interaction the execution harness. We focus on two editable components (Figure 1): reusable textual skills S that guide actions and context-construction code C that selects and presents observed evidence. Model parameters and the execution interface remain fixed during harness adaptation.

An initial harness encodes assumptions about how tasks are expressed and what evidence is available during execution. Changes in instructions or visual conditions can make its action guidance or evidence selection inadequate. Revising the harness still requires task-specific experience, which can be costly to collect in physical environments (Kim et al., 2024; Tang et al., 2025). Prior work on reflection and experience-guided learning shows that such experience can be retained as reusable guidance (Shinn et al., 2023; Zhao et al., 2024). The challenge is therefore to turn limited execution evidence into revisions that improve the harness on later tasks, rather than corrections confined to the episode that produced it.

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.

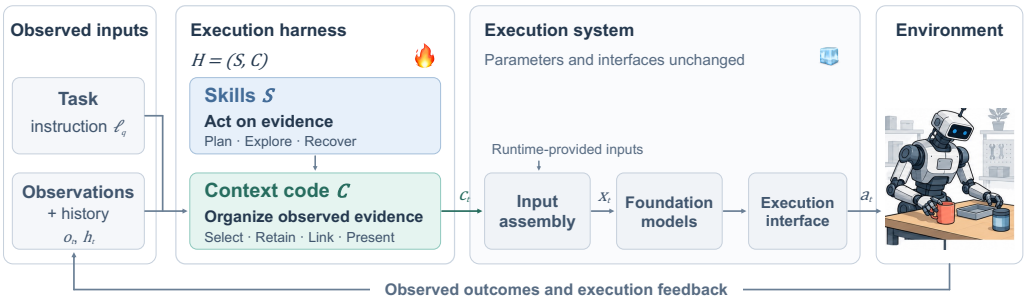


Figure 1: Execution harness for embodied agents. The editable harness $H = (S, C)$ comprises textual skills S for action guidance and context-construction code C for organizing observed evidence. The fixed runtime assembles the complete model input x_t from constructed context c_t and any additional runtime-provided inputs; model parameters and runtime interfaces remain unchanged during harness evolution.

Feedback-driven adaptation already spans reusable skill optimization (Yang et al., 2026; Ju et al., 2026) and executable harness evolution (Lee et al., 2026; Lin et al., 2026), including skill-and-context adaptation for embodied agents (Wang et al., 2026b). Useful revisions, however, require more than a task-level success signal. A failure may reflect feedback omitted from the actual model input, an inadequate response to available feedback, or both. This motivates examining what evidence reached each decision and how the agent responded. At the same time, a proposed skill revision may depend on observed evidence not yet presented to the model. Evaluating that revision together with the corresponding change to context construction allows their combined effect to be assessed as one harness candidate, rather than treating the two edits as independent improvements.

We introduce EnactOpt, a framework for diagnosis-guided joint revision of textual skills and context-construction code (Figure 2). It diagnoses adjacent decisions in training rollouts, relating action feedback to the actual model input and the agent's response. Structured episode summaries consolidate the reported evidence, uncertainty, and repair and preservation suggestions into feedback for a single LLM optimizer. Given the source harness and this feedback, the optimizer uses prompt-guided edit operators to organize targeted changes to S, C, or both within one harness candidate. Interface and runtime compatibility checks screen candidates before validation-based pool retention. The retained versions supply subsequent iterations, and the final validation-selected harness is fixed for testing.

We evaluate benchmark-specific EnactOpt harnesses on VLABench (Zhang et al., 2025b), ALFWorld (Shridhar et al., 2020), and Embodied Agent Interface (EAI) (Li et al., 2024), spanning low-level policy execution and high-level action interfaces. Under the reported configurations, EnactOpt improves task success by approximately 10–27 percentage points over the initial harnesses and outperforms the evaluated GEPA (Agrawal et al., 2026), TextGrad (Yuksekgonul et al., 2024), and Trace2Skill (Ni et al., 2026) baselines on all three benchmarks (Table 1). The gains extend to within-benchmark task conditions not used for evolution or candidate selection. On VLABench, the ablation comparisons show larger performance gaps under shifted conditions than in-distribution. Joint revision also achieves higher generalization success with fewer optimization passes than separate skill and code updates (Table 2).

Our contributions are threefold:

- We introduce EnactOpt, a framework that treats textual skills and context-construction code as a coupled adaptation target. A single proposal can revise either or both components, enabling their combined behavior to be evaluated while models and runtime remain fixed.
- We develop a diagnosis-to-revision feedback interface that relates action feedback to actual model inputs and subsequent responses. Structured episode summaries convey reported evidence and uncertainty, while prompt-guided edit operators organize targeted changes and preservation requests.

- We report gains over initial harnesses and evaluated baselines across three embodied benchmarks, including gains beyond the evolution conditions. VLABench ablations show that comparable in-distribution success can mask differences in generalization, and that joint revision attains higher generalization success with fewer optimization passes than separate updates.

2 Related Work

2.1 Embodied Agent Adaptation

Adaptation in embodied agents spans adjustments during execution and updates retained for later tasks. Hierarchical models and code-based policies connect task-level instructions to executable behavior (Shi et al., 2025; Liang et al., 2023), while closed-loop systems use observations and execution feedback to revise plans and recover from failures (Huang et al., 2023; Zhi et al., 2025). Beyond these within-task adjustments, policy parameters can be adapted through demonstration-based fine-tuning (Kim et al., 2024) or reinforcement learning from deployment experience (Physical Intelligence et al., 2025a). Experience can also be retained outside model parameters: DROC distills human language corrections into retrievable knowledge (Zha et al., 2024), whereas ASPIRE autonomously refines robot control programs and consolidates validated repairs into a reusable skill library (Lu et al., 2026). EnactOpt follows this latter direction by jointly revising reusable textual skills and context-construction code while keeping the underlying models fixed.

2.2 Agent Harness Optimization

Feedback-driven optimization differs in both the artifact revised and how experience is consolidated. ExpeL and Voyager retain experience as reusable natural-language guidance and executable skills, respectively (Zhao et al., 2024; Wang et al., 2023). TextGrad propagates textual feedback through computation graphs, while GEPA evolves prompts through trajectory-based reflection (Yuksekgonul et al., 2024; Agrawal et al., 2026). For reusable guidance, ACE incrementally updates structured playbooks, whereas Trace2Skill consolidates lessons from a pool of trajectories into skills (Zhang et al., 2026; Ni et al., 2026). Skill revision further addresses update control and attribution through bounded, validation-gated edits (Yang et al., 2026), distinctions between skill defects and execution lapses (Ju et al., 2026), and preservation-aware structural reorganization informed by accumulated diagnoses and success–failure contrasts (Wang et al., 2026a).

For executable harnesses, AFlow formulates workflow construction as search over code (Zhang et al., 2025a). A central distinction is the evidence exposed to the optimizer: Meta-Harness provides access to prior code, scores, and raw trajectories, whereas AHE organizes experience into a layered evidence corpus and links edits to predicted effects through change manifests (Lee et al., 2026; Lin et al., 2026). Selection also depends on the available supervision: RHO uses retrospective self-validation and self-consistency followed by pairwise self-preference (Pan et al., 2026), while HarnessEvolve diagnoses failures against verified, answer-conditioned reference trajectories and gates updates by quality and task performance (Jiang et al., 2026).

In embodied settings, Zetta revises critics, recovery strategies, and tools (Ding et al., 2026), whereas SHAPER evolves skills and context-construction code with context-aware diagnosis and prompts requesting local edits and preservation of successful behavior (Wang et al., 2026b). EnactOpt diagnoses adjacent decisions by relating action feedback to the actual model input and action at the next decision, while limiting editor-visible execution feedback to structured summaries without raw-trajectory retrieval. Unlike SHAPER's reported skill-then-code schedule, EnactOpt permits either or both components to change in one proposal, with prompts specifying six edit operators and requesting supporting evidence and behavior to preserve.

3 Problem Setup

3.1 Embodied Task Setting

We consider an embodied task q specified by an environment $\mathcal{E}_q$ with initial conditions, an instruction ℓ_q, and a task-level success criterion. At decision step t, the agent selects an action a_t using

information drawn from ℓ_q, the current observation o_t, and the available interaction history h_t of prior observations, actions, and feedback. Executing a_t yields the next observation o_{t+1} and any available execution feedback f_{t+1}; interaction continues until termination. The agent is not assumed to observe the full environment state or information reserved for task evaluation.

Let $\mathcal{M}$ denote the fixed model configuration used for task execution and $\mathcal{R}$ the non-editable runtime responsible for model-input assembly, model invocation, output parsing, and action execution. Depending on the benchmark, $\mathcal{M}$ may include a reasoning model coupled to a low-level policy, or a language model that directly issues commands through a high-level action interface. The index t counts agent decisions rather than individual control steps, so one action a_t may invoke a low-level policy for multiple control steps. Both $\mathcal{M}$ and $\mathcal{R}$ remain fixed during harness evolution; benchmark-specific configurations are described in Section 5.1.

3.2 Execution Harness

We represent the editable part of the execution harness as

$$H = (S, C), \tag{1}$$

where S contains reusable textual skills that guide task execution, exploration, and failure recovery. The executable context-construction code C selects, retains, links, and presents information derived from available observations and interaction history. It processes observed evidence rather than providing access to unobserved environment state. Harness adaptation is restricted to S and C; the model configuration $\mathcal{M}$ and runtime $\mathcal{R}$ remain fixed.

At decision step t, we write the context produced by C schematically as

$$c_t = C(\ell_q, S, o_t, h_t).$$

The fixed runtime $\mathcal{R}$ assembles c_t with any additional runtime-provided observations or messages to form x_t, the complete input delivered to the decision-making model. Consequently, information omitted from c_t may still be present in x_t. Whether feedback was presented to the model must therefore be assessed against x_t, rather than the constructed context alone.

The artifacts S and C remain fixed throughout each rollout, whereas c_t and x_t are constructed at each decision. For a task q, the fixed execution system $(\mathcal{M}, \mathcal{R})$ and harness H induce a rollout

$$\xi \sim \mathrm{Rollout}(\mathcal{E}_q, \ell_q; \mathcal{M}, \mathcal{R}, H), \tag{2}$$

where ξ denotes the recorded episode, including decision-level observations, actual model inputs, actions, and returned feedback, together with the task outcome reported at termination. How these records are converted into feedback for the optimizer is specified in Section 4.2.

3.3 Harness Adaptation Objective

With the model configuration $\mathcal{M}$ and runtime $\mathcal{R}$ fixed, adaptation optimizes only the harness $H = (S, C)$. Let $\mathcal{H}$ denote the space of admissible harnesses that respect the component boundaries and runtime-interface requirements defined above. Let p_{target} be the target task distribution and $R_q(\xi)$ the task-level score assigned to a rollout ξ of task q, with larger values indicating better performance.

Our objective is to maximize the expected task score under p_{target}:

$$\begin{aligned} J(H) &= \mathbb{E}_{q \sim p_{\mathrm{target}}} \mathbb{E}_{\xi \sim \mathrm{Rollout}(\mathcal{E}_q, \ell_q; \mathcal{M}, \mathcal{R}, H)} \big[R_q(\xi) \big], \\ H^\star &\in \arg\max_{H \in \mathcal{H}} J(H). \end{aligned} \tag{3}$$

For binary task-success scores, $J(H)$ is the expected success rate over the target task distribution. Here, $H^\star$ denotes an ideal optimal harness rather than the output guaranteed by the adaptation procedure.

In practice, $J(H)$ is not directly observed during adaptation. EnactOpt proposes revisions using finite execution feedback collected on training tasks $\mathcal{D}_{\mathrm{train}}$ and selects candidates according to empirical performance on validation tasks $\mathcal{D}_{\mathrm{val}}$. The target distribution may include task conditions absent from both adaptation sets. Validation therefore provides a criterion for candidate selection, while generalization beyond the adaptation conditions is assessed separately.

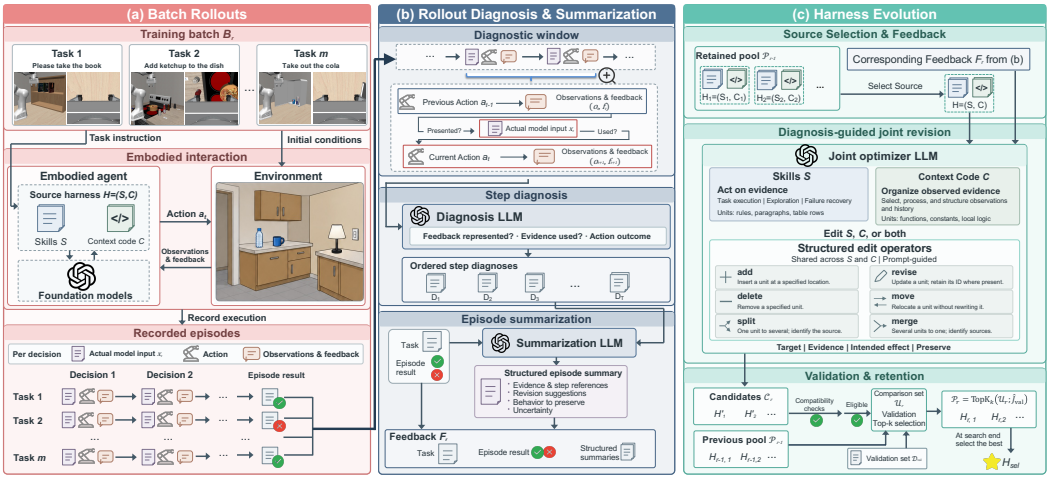


Figure 2: Overview of EnactOpt. (a) A source harness $H = (S, C)$ executes training batch B_r, recording actual model inputs, actions, observations, and episode outcomes. (b) Adjacent-decision diagnoses and episode summaries form its feedback F_r. (c) A joint optimizer uses the same H and F_r to propose candidates $\mathcal{C}_r$ through prompt-guided structured edit operators. Compatible candidates are compared with the previous pool $\mathcal{P}_{r-1}$ on validation tasks to form $\mathcal{P}_r$. Scenes are illustrative, not experimental results.

4 EnactOpt: Diagnosis-Guided Harness Evolution

4.1 Evolution Overview

Figure 2 summarizes EnactOpt's iterative evolution loop. Starting from $H_0 = (S_0, C_0)$, EnactOpt maintains a pool of retained harnesses while keeping $\mathcal{M}$ and $\mathcal{R}$ fixed. At evolution iteration r, a source harness $H = (S, C)$ is selected from $\mathcal{P}_{r-1}$ to execute a batch B_r from the training set $\mathcal{D}_{\text{train}}$, producing rollouts and their native execution outcomes.

Each rollout is analyzed through adjacent-decision diagnoses, which are aggregated into a structured episode summary. The task descriptions, native outcomes, and complete summaries form the feedback F_r associated with the source harness. A single joint optimizer receives this same H together with F_r to propose candidate harnesses $\mathcal{C}_r$, each comprising complete skill and context-code files. Either component may remain unchanged in a proposal.

Candidates undergo local interface and runtime compatibility checks before evaluation on $\mathcal{D}_{\text{val}}$. Validation-based selection updates the retained pool $\mathcal{P}_r$, which supplies source versions for subsequent iterations. At the end of the search, a harness H_{sel} is selected by validation performance and fixed for test evaluation.

4.2 Execution Diagnosis and Feedback Construction

Adjacent-decision diagnosis. Task-level outcomes alone do not reveal whether relevant feedback was missing from the model input or was available but inadequately used. We therefore examine adjacent decisions (Figure 2(b)). At decision t, the diagnostic window relates the preceding action a_{t-1} and its returned observation and feedback (o_t, f_t) to the current model input x_t and action a_t. It also includes the observation and feedback (o_{t+1}, f_{t+1}) returned by a_t to assess the current action's outcome. The first step has no preceding action feedback; the final step remains assessable because no subsequent decision is required.

The diagnostic LLM produces a report D_t assessing whether preceding feedback is faithfully presented in x_t, whether the current action reflects appropriate use of that feedback, and what outcome the action produces. The presentation check uses the full input x_t, not c_t alone, because the runtime may append observations or messages. Missing or misrepresented evidence may suggest a revision to C; an inadequate response to available evidence may suggest a revision to S. These are nonex-

clusive revision hypotheses, not causal proofs. Ambiguous or incomplete evidence is retained as uncertainty.

Structured episode feedback. The episode summarizer receives the task description, native episode outcome, and all ordered diagnoses $D_1, \ldots, D_T$, where T is the number of decisions; it does not receive the raw trajectory. Its prompt requests a structured summary retaining concrete conditions, observations and explanations reported by the diagnoses, and step references for supporting and conflicting evidence. It also requests repair suggestions, behavior to preserve, and unresolved uncertainty, so that proposed changes account for useful existing behavior as well as observed problems. The summarized evidence remains diagnosis-mediated: step references do not independently verify an interpretation or guarantee lossless summarization.

For a batch B_r executed by the source harness H, the task descriptions, native outcomes, and complete episode summaries form F_r. The optimizer receives H and F_r without attached raw-trajectory windows, actor replies, or context-history excerpts. Step references indicate the evidence reported in the diagnoses, but do not trigger raw-log retrieval. Raw logs remain available locally for inspection, outside the optimizer's feedback interface. This feedback guides the joint revision of S and C in Section 4.3.

4.3 Joint Revision with Structured Edit Operators

Given $H = (S, C)$ and its batch feedback F_r, one joint optimizer LLM proposes complete replacement files for both components:

$$H' = (S', C') = \mathrm{Optimizer}(S, C, F_r). \tag{4}$$

Each proposal yields a candidate $H' \in \mathcal{C}_r$, with either component allowed to remain unchanged. Context-construction code selects, processes, and organizes existing observations and interaction history into task-relevant context, including structured state representations. Skills guide task execution, further exploration, and failure recovery. Both affect action selection; new environment observations arise through agent actions or tool calls, not direct environment interaction by the context builder.

The optimizer is prompted to use six structured edit operators: add, revise, delete, move, split, and merge. Editable units are Markdown items, paragraphs, or table rows in skills, and functions, constants, or local logic in context-construction code. For each proposed edit, the optimizer is prompted to identify its target in the source harness, the selected operator, supporting feedback, the intended effect, and behavior to preserve. These prompt-level conventions guide incremental revision while the optimizer returns complete files; preservation of untouched content is requested rather than programmatically enforced.

4.4 Validation and Harness Retention

Each candidate in $\mathcal{C}_r$ first undergoes local interface and runtime compatibility checks under the fixed execution system. Only compatible candidates proceed to evaluation on $\mathcal{D}_{\mathrm{val}}$. Their empirical validation score $\hat{J}_{\mathrm{val}}(H)$ is computed from the task-level scores $R_q(\xi)$; for binary outcomes, it is the validation success rate. Compatibility checks determine eligibility for evaluation, not behavioral preservation or improved task performance.

Let $\mathcal{U}_r$ denote the comparison set comprising compatible new candidates and the previously retained harnesses $\mathcal{P}_{r-1}$. Following the top-k retention scheme in Figure 2(c), these versions are ranked by $\hat{J}_{\mathrm{val}}$, and up to k harnesses are retained:

$$\mathcal{P}_r = \mathrm{TopK}_k\left(\mathcal{U}_r;\, \hat{J}_{\mathrm{val}}\right), \tag{5}$$

where k is the pool capacity and TopK_k selects the highest-scoring versions. Thus, a new proposal competes with existing versions rather than automatically replacing its source.

The updated pool $\mathcal{P}_r$ supplies source harnesses for subsequent iterations under the experiment's search schedule. At the end of the search, the retained harness with the highest validation score is selected as H_{sel} and fixed for test evaluation. Training feedback thus guides proposal generation, whereas validation performance guides retention and final selection; performance beyond the adaptation conditions is assessed separately at test time.

Table 1: Task success rates (%, higher is better) across three benchmarks. Bold indicates the best result in each column. Colored annotations show percentage-point differences from Naive (green: positive; red: negative). † denotes our skill-only implementations of TextGrad and Trace2Skill.

Method	VLABench	ALFWorld	EAI
Reference Systems			
Naive	40.87	51.46	68.09
H_0	47.39 +6.52	66.06 +14.60	72.34 +4.26
Adaptation Methods			
TextGrad†	36.09 -4.78	86.50 +35.04	71.63 +3.55
Trace2Skill†	40.54 -0.33	78.83 +27.37	72.70 +4.61
GEPA	44.46 +3.59	83.21 +31.75	86.88 +18.79
EnactOpt	**57.83 +16.96**	**92.70 +41.24**	**95.74 +27.66**

5 Experiments

5.1 Experimental Setup

Benchmarks and protocol. We evaluate separately evolved harnesses on VLABench (Zhang et al., 2025b), ALFWorld (Shridhar et al., 2020), and EAI's VirtualHome Action Sequencing (Li et al., 2024). All MLLM components use GPT-5.4 mini (OpenAI, 2026); VLABench additionally uses the benchmark-released π_0 policy (Black et al., 2024), fixed during evolution. ALFWorld and EAI use high-level action interfaces. Train/validation/test sizes are 30/30/920 for VLABench and 30/30/282 for EAI. For VLABench, adaptation and candidate selection use the in-distribution track (T1); testing covers T1, commonsense-based target selection (T3), context-rich instruction understanding (T4), and unseen textures (T6). GEPA and EnactOpt share a frozen ALFWorld 30/30/274 split, with 140 Seen and 134 Unseen test instances. GEPA's unscored Unseen instance is counted as a failure.

Evolution settings. Motivated by the cost of collecting embodied interaction data, we restrict EnactOpt to 30 training task instances per benchmark. These tasks form three fixed, disjoint batches of ten, with one batch per evolution iteration, yielding one training epoch. Here, an epoch counts task coverage, not the total rollouts collected by different source harnesses. Candidates passing compatibility checks are evaluated on the full 30-instance validation set. The default pool capacity is $k = 2$: after each iteration, the top two harnesses among the new candidates and the previous pool are retained by validation success rate. After the third iteration, the retained harness with the highest validation success rate is selected and fixed for test evaluation.

Baselines and metrics. Naive uses each benchmark's official pipeline without our additional harness; H_0 is the initial harness without adaptation. TextGrad (Yuksekgonul et al., 2024) and Trace2Skill (Ni et al., 2026) are our skill-only implementations, with context-construction code kept fixed. GEPA (Agrawal et al., 2026) and EnactOpt start from H_0. We report task success rates (%) and differences in percentage points. These comparisons are not matched-compute evaluations. Benchmark-specific data protocols are described in Appendix A.

Optimization cost. For the VLABench ablations, we count complete optimization passes: candidate generation, evaluation on the full validation set, and pool retention for one batch. EnactOpt and the diagnosis and edit-operator ablations use one pass per batch; separate S/C revision uses two, one per component. The three-batch schedule therefore requires three versus six passes, normalized to $1\times$ and $2\times$ relative to EnactOpt; H_0 is not optimized.

5.2 Main Results

With 30 training task instances per benchmark, EnactOpt improves on the initial harness across both low-level policy execution and high-level action interfaces (Table 1). Its gains over H_0 are 10.43, 26.64, and 23.40 percentage points on VLABench, ALFWorld, and EAI, respectively. These gains extend beyond the benefit of constructing H_0: execution experience supports further refinement of an already configured agent without changing its underlying models.

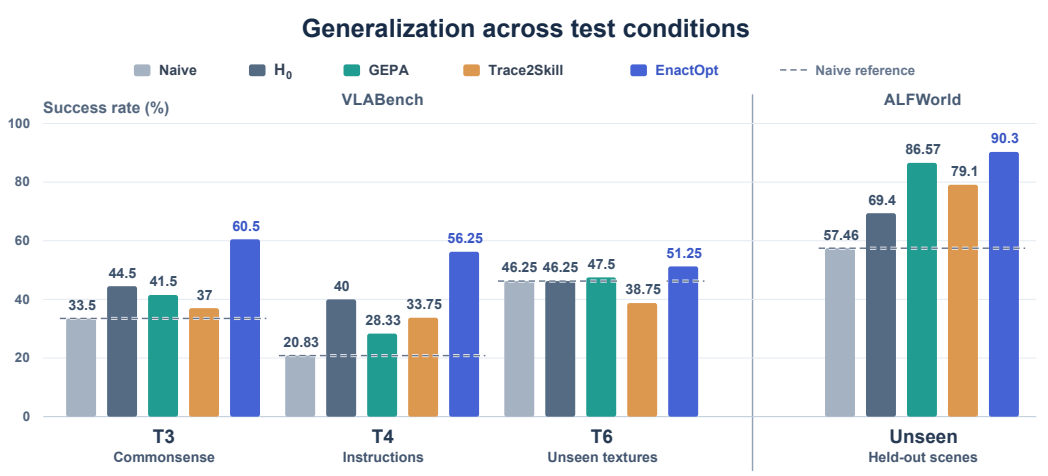


Figure 3: Generalization on VLABench T3/T4/T6 and ALFWorld Unseen. Bars show task success rates (%); horizontal dashed lines denote the Naive baseline. Adaptation is performed separately for each benchmark. The Trace2Skill VLABench and EnactOpt ALFWorld results use different runs from the corresponding entries in Table 1.

EnactOpt also outperforms every evaluated adaptation baseline, but the relative strengths of those baselines vary by benchmark. Our skill-only TextGrad implementation is the strongest adaptation baseline on ALFWorld, whereas GEPA leads the alternatives on VLABench and EAI. On VLABench, however, even GEPA's adaptation from H_0 falls below the unchanged H_0 reference. Thus, the ability to revise model-external artifacts does not by itself ensure a better harness. These complete-pipeline comparisons motivate examining the diagnostic and revision choices in Section 5.4, rather than attributing the gains to editable components alone.

5.3 Generalization across Test Conditions

Adaptation on VLABench T1 transfers to other test conditions, but the gains vary with the type of shift (Figure 3). Although evolution and candidate selection use only T1, EnactOpt improves T3 and T4 over H_0 by 16.00 and 16.25 percentage points, compared with 5.00 points on T6; it also exceeds GEPA on all three tracks. The revisions thus remain useful beyond the adaptation conditions, with the largest improvements on the tracks that change how the target or request must be interpreted, rather than visual appearance.

ALFWorld complements this comparison with a change in scene rather than evaluation track. On Unseen scenes, EnactOpt exceeds both H_0 and GEPA, with a 3.73-percentage-point advantage over GEPA in the runs shown. The advantage therefore persists against an adapted baseline, not only the initial harness. Unseen retains the same task families; both benchmarks assess within-benchmark transfer without test-time harness revision, rather than transfer of one harness across benchmarks.

5.4 Ablation Studies

We examine diagnostic evidence, structured edit operators, and joint revision on VLABench (Table 2). *w/o Adjacent-Decision Diagnosis* retains joint revision and edit operators but uses only task descriptions and terminal outcomes as diagnostic evidence. Relative to H_0, it gains 3.75 percentage points on In-Distribution but loses 1.91 points on Generalization, whereas EnactOpt improves both groups. This contrast shows why in-distribution improvement alone is not a sufficient measure of a reusable revision: the gain from outcome-only feedback does not extend to the shifted test conditions.

w/o Structured Edit Operators retains diagnosis and joint revision but removes prompt-level editing conventions. Its success rates fall below H_0 in both groups, by 2.08 percentage points In-Distribution and 2.50 points on Generalization. Retaining diagnostic feedback is therefore not sufficient to obtain

Table 2: VLABench ablations on T1 (In-Distribution) and T3/T4/T6 (Generalization). Rates (%); subscripts: percentage-point changes from H_0. Cost: optimization passes relative to EnactOpt.

Variant	In-Distribution	Generalization	Optimization cost
H_0	58.33	43.53	–
w/o Adjacent-Decision Diagnosis	62.08 +3.75	41.62 -1.91	1×
w/o Structured Edit Operators	56.25 -2.08	41.03 -2.50	1×
w/o Joint Revision	63.33 +5.00	50.88 +7.35	2×
EnactOpt	**63.75 +5.42**	**55.74 +12.21**	1×

useful revisions in this comparison. The result points to the importance of how the optimizer uses that evidence to organize edits, not only whether it receives the evidence.

w/o Joint Revision still edits both S and C, but separately. Its In-Distribution success remains close to EnactOpt (63.33% vs. 63.75%), whereas Generalization is lower (50.88% vs. 55.74%), despite using twice as many optimization passes. The comparison concerns how revisions are proposed and evaluated, not whether both components can change. Additional component-wise passes do not recover the generalization performance of joint revision in these runs.

6 Discussion

The results point to a distinction between the capabilities of an underlying model and the way an agent uses those capabilities. Skills and context-construction code can change how observed evidence is organized and acted upon while model parameters and runtime interfaces remain fixed. This makes the execution harness a complementary adaptation target, rather than a substitute for model or policy training. Improvements at this level do not demonstrate new perception or low-level control capabilities.

The harness artifacts in Appendix B show concrete changes in memory use, recovery, and controller-facing instructions. ALFWorld expands the retained-history window from two to 50 turns and revises search and responses to ineffective actions. EAI replaces stopping after one correction with continued attempts within budget, while emphasizing current state and the latest feedback. VLABench explicitly presents observed entity names and prescribes short, canonical commands that preserve their exact spelling. These are observed differences between harnesses, not isolated attributions of the performance gains.

Our evaluation focuses on within-benchmark adaptation. Future work could investigate the transfer of evolved harnesses across benchmarks, their application to broader task compositions, and their reliability on physical robots. Controlled, matched-budget studies on independent test data would help disentangle the contributions of skill revision, context-code revision, and diagnostic feedback. Measuring both evolution and execution costs would also clarify the trade-off between task performance and overall resource use.

7 Conclusion

We presented EnactOpt, which uses execution diagnosis to jointly evolve skills and context-construction code while keeping the underlying models fixed. Evaluations on VLABench, ALFWorld, and EAI show improved task success, with gains also observed under the evaluated within-benchmark shifts. The VLABench ablations show that variants with similar in-distribution success can generalize differently; joint revision also achieves higher generalization success with fewer optimization passes than separate updates. These findings motivate treating evidence presentation and action guidance as a coupled adaptation target, and assessing harness revisions by whether they remain useful beyond the conditions used for evolution.

AI USE STATEMENT

Generative AI tools, including GitHub Copilot, assisted with manuscript drafting and language polishing, literature lookup and citation formatting, LaTeX tables, and code for figure preparation. Model use in the evaluated agents is described in Section 5.1. The authors retain responsibility for the final text, claims, and research artifacts.

ETHICS STATEMENT

Our experiments use official VLABench, ALFWorld, and EAI benchmark resources in simulated or text-based environments. The results do not establish safe operation on physical robots; deploying evolved execution code would require additional safety validation and oversight.

REPRODUCIBILITY STATEMENT

Section 4 describes the harness evolution procedure. Section 5.1 and Appendix A document the evaluated task subsets, data partitions, and training-batch construction. Appendix B provides the evolved skill texts and context-construction code for each benchmark.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2507.19457v2.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025.

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

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

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, 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. *arXiv preprint arXiv:2303.03378*, 2023. URL https://arxiv.org/abs/2303.03378.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner Monologue: Embodied reasoning through planning with language models. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Wen Jiang, Mingmin Chu, Yimeng Tian, Qianxin Zhang, Haofei Yang, Rui Yang, Yang Liu, Tao Lv, and Fangming Li. HarnessEvolve: Learning from reference trajectories for reliable agent self-evolution. *arXiv preprint arXiv:2609.00829*, 2026. URL https://arxiv.org/abs/2609.00829v1.

Ruofei Ju, Xinrui Wang, Xin Ding, Yifan Yang, Hao Wu, Shiqi Jiang, Qianxi Zhang, Hao Wen, Xiangyu Li, Weijun Wang, Kun Li, Yunxin Liu, Haipeng Dai, Wei Wang, and Ting Cao. EmbodiSkill: Skill-aware reflection for self-evolving embodied agents, 2026. URL https://arxiv.org/abs/2605.10332v2.

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

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses, 2026. URL https://arxiv.org/abs/2603.28052.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li Erran Li, Ruohan Zhang, et al. Embodied Agent Interface: Benchmarking LLMs for embodied decision making. In *Advances in Neural Information Processing Systems*, volume 37, pp. 100428–100534. Curran Associates, Inc., 2024. doi: 10.52202/079017-3188. URL https://doi.org/10.52202/079017-3188.

Qixiu Li, Yu Deng, Yaobo Liang, Lin Luo, Lei Zhou, Chengtang Yao, Lingqi Zeng, Zhiyuan Feng, Huizhi Liang, Sicheng Xu, et al. Scalable vision-language-action model pretraining for robotic manipulation with real-life human activity videos. *arXiv preprint arXiv:2510.21571*, 2025.

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 *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500. IEEE, 2023. doi: 10.1109/ICRA48891.2023.10160591. URL https://doi.org/10.1109/ICRA48891.2023.10160591.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses, 2026. URL https://arxiv.org/abs/2604.25850v4.

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*, 2026. URL https://arxiv.org/abs/2607.00272v1.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2Skill: Distill trajectory-local lessons into transferable agent skills, 2026. URL https://arxiv.org/abs/2603.25158.

OpenAI. GPT-5.4 Thinking System Card. https://openai.com/index/gpt-5-4-thinking-system-card/, March 2026.

Wenbo Pan, Shujie Liu, Chin-Yew Lin, Jingying Zeng, Xianfeng Tang, Xiangyang Zhou, Yan Lu, and Xiaohua Jia. Evolving agents in the dark: Retrospective harness optimization via self-preference. *arXiv preprint arXiv:2606.05922*, 2026. URL https://arxiv.org/abs/2606.05922v3.

Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, et al. $\pi^*_{0.6}$: a VLA that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025a. URL https://arxiv.org/abs/2511.14759.

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

Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, et al. Hi Robot: Open-ended instruction following with hierarchical vision-language-action models. *arXiv preprint arXiv:2502.19417*, 2025. URL https://arxiv.org/abs/2502.19417.

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*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://doi.org/10.52202/075280-0377.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020. URL https://arxiv.org/abs/2010.03768.

Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep reinforcement learning for robotics: A survey of real-world successes. *Annual Review of Control, Robotics, and Autonomous Systems*, 8(1):153–188, 2025. doi: 10.1146/annurev-control-030323-022510. URL https://doi.org/10.1146/annurev-control-030323-022510.

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.

Hanyu Wang, Yifan Lan, Bochuan Cao, Lu Lin, and Jinghui Chen. SkillGrad: Optimizing agent skills like gradient descent. *arXiv preprint arXiv:2605.27760*, 2026a. URL https://arxiv.org/abs/2605.27760v1.

Peidong Wang, Zhiming Ma, Ying Chang, Xufang Luo, Yiqun Zhang, Zihan Wang, Xiaocui Yang, Shi Feng, Yuqing Yang, and Dongsheng Li. Self-evolving embodied agents via skill-harness evolution, 2026b. URL https://arxiv.org/abs/2608.11350v2.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills, 2026. URL https://arxiv.org/abs/2605.23904v2.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024. URL https://arxiv.org/abs/2406.07496.

Lihan Zha, Yuchen Cui, Li-Heng Lin, Minae Kwon, Montserrat Gonzalez Arenas, Andy Zeng, Fei Xia, and Dorsa Sadigh. Distilling and retrieving generalizable knowledge for robot manipulation via language corrections. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 15172–15179. IEEE, 2024. doi: 10.1109/ICRA57147.2024.10610455. URL https://doi.org/10.1109/ICRA57147.2024.10610455.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=z5uVAKwmjf.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.04618v3.

Shiduo Zhang, Zhe Xu, Peiju Liu, Xiaopeng Yu, Yuan Li, Qinghui Gao, Zhaoye Fei, Zhangyue Yin, Zuxuan Wu, Yu-Gang Jiang, et al. VLABench: A large-scale benchmark for language-conditioned robotics manipulation with long-horizon reasoning tasks. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11142–11152. IEEE, 2025b. doi: 10.1109/ICCV51701.2025.01037. URL https://doi.org/10.1109/ICCV51701.2025.01037.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://doi.org/10.1609/aaai.v38i17.29936.

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. doi: 10.1109/ICRA55743.2025.11127975. URL https://doi.org/10.1109/ICRA55743.2025.11127975.

APPENDIX

CONTENTS

A EnactOpt Implementation and Evaluation Details **15**

A.1 VLABench Protocol 15

A.2 ALFWorld Protocol 18

A.3 EAI Protocol 18

B Benchmark Harness Artifacts **20**

B.1 VLABench 20

B.2 ALFWorld 27

B.3 EAI 29

A EnactOpt Implementation and Evaluation Details

This appendix records implementation and evaluation details.

A.1 VLABench Protocol

We use six primitive tasks: `select_chemistry_tube`, `select_poker`, `select_drink`, `add_condiment`, `select_book`, and `select_toy`. The VLABench-released `pi0-primitive-10task` checkpoint[1] remains frozen during harness evolution. Task selection was informed by the checkpoint's reported T1 performance and T1–T4 performance gaps, rather than random sampling of primitive tasks.

We use the official T1 (In-Distribution), T3 (Common Sense), T4 (Semantic Instruction), and T6 (Unseen Texture) tracks.[2] T3 and T4 concern commonsense target identification and context-rich instruction understanding, respectively; their reasoning demands can overlap. T6 changes textures, materials, and backgrounds; it is not a texture-only, single-factor intervention. Figures 4 and 5 illustrate these tracks with initial observations and task instructions.

We partition the official task–track episode lists using the fixed indices in Table 3. Train30 and Val30 use T1 only. Test920 contains 240, 200, 240, and 240 episodes from T1, T3, T4, and T6, respectively. T3 excludes `select_poker`; T2 is not included.

Table 3: Fixed VLABench episode splits. Indices are zero-based and inclusive within each official task–track list. Counts include only valid selected task–track pairs.

Use	Track	Indices	Episodes
Harness evolution (Train30)	T1	0–4	30
Candidate selection (Val30)	T1	5–9	30
Test	T1, T3, T4, T6	10–49	920

For EnactOpt, Train30 is partitioned into three fixed, disjoint Train10 batches, each containing all six tasks with one or two episodes per task. Their union is Train30, but this membership partition does not bound the total number of training rollouts. Val30 is used for candidate selection.

Episode identities remain fixed across iterations, but simulator realizations and policy actions need not be deterministic. Exclusion from harness evolution does not establish exclusion from policy pretraining or prior fine-tuning.

[1] `https://huggingface.co/VLABench/pi0-primitive-10task`

[2] `https://github.com/OpenMOSS/VLABench#evaluate`

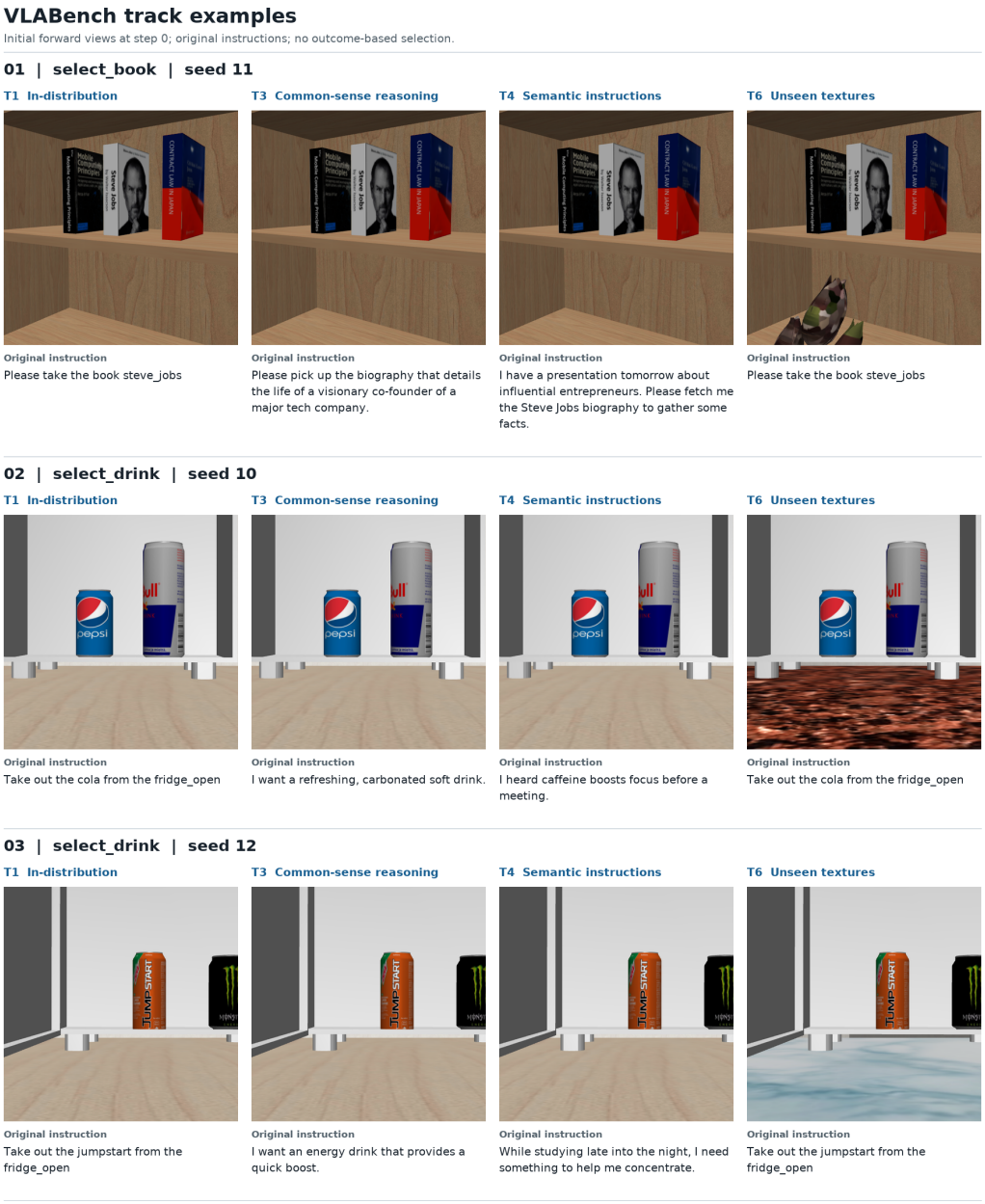


Figure 4: VLABench track examples for book and drink selection. Each row compares initial forward views and task instructions across T1, T3, T4, and T6. T3/T4 illustrate changes in the instruction, while T6 illustrates changes in the visual setting. The images show initial states, not execution outcomes.

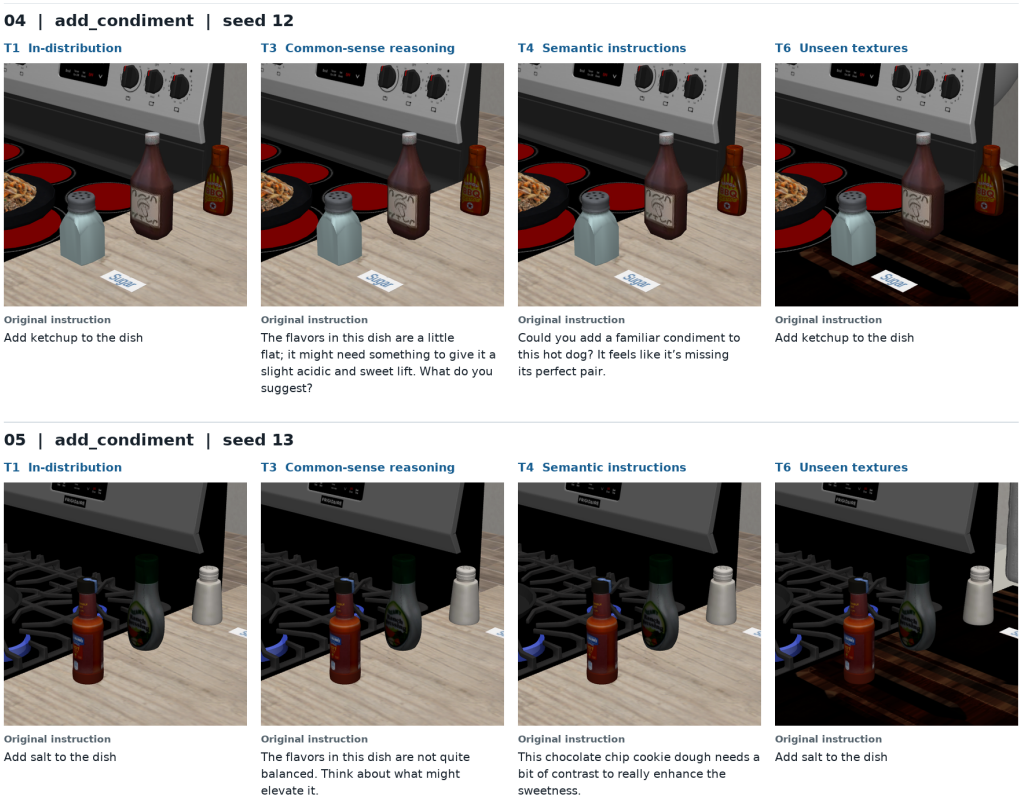


Figure 5: VLABench track examples for condiment addition. The columns follow the same T1/T3/T4/T6 order as Figure 4, showing initial forward views and task instructions for two further episodes.

A.2 ALFWORLD PROTOCOL

We use ALFWorld 0.4.2 with `AlfredTWEnv` and all six supported task families. Following the text-environment loader, we exclude unsupported movable and sliced variants and require valid game files and `solvable=true`. The filtered official pools contain 3,553 training, 140 Seen, and 134 Unseen instances. Eligibility uses dataset annotations, not screening by agent success.

Train30 and Val30 are selected from the official training pool. We hash-order its 108 scenes with seed 42 and divide them into two disjoint pools of 54 scenes. Within each pool, game identifiers are hash-ordered separately for each task family using the same seed, and the first five instances per family are selected (Table 4). This deterministic, task-stratified procedure yields 30 instances per partition. Train30 and Val30 span 23 and 20 scenes, respectively, share neither instances nor scenes, and remain fixed throughout harness evolution.

Table 4: Task-family counts for the fixed ALFWorld Train30/Val30 split and the complete Seen/Unseen evaluation sets.

Task family	Train30	Val30	Seen	Unseen
Pick & Place	5	5	35	24
Pick Two & Place	5	5	24	17
Examine in Light	5	5	13	18
Clean & Place	5	5	27	31
Heat & Place	5	5	16	23
Cool & Place	5	5	25	21
Total	30	30	140	134

We partition the fixed Train30 into three disjoint Train10 batches by task-stratified shuffling with `random.Random(42)`. Each batch contains one or two instances per family, and the three batches cover Train30 exactly once. Training rollouts supply harness-revision feedback without updating model parameters; Val30 is used for candidate selection.

The selected harness is fixed for evaluation on all 140 official `valid_seen` and 134 `valid_unseen` instances, without further subsampling. These sets are instance-disjoint from Train30 and Val30, but Seen shares some scenes with the development sets. Unseen denotes scenes absent from the official training pool.

A.3 EAI PROTOCOL

We use all 342 VirtualHome Action Sequencing instances in `scene_1`, covering 26 official activity categories. All partitions come from this same scene; no instances are excluded based on model performance.

We form indivisible groups by taking the transitive closure of two relations: a shared numeric identifier prefix, or equivalence of symbolic initial-state graphs and public goal specifications under attributed directed-graph isomorphism. The comparison uses the full symbolic graphs with geometric attributes removed; reference action sequences are not used. This yields 310 groups, comprising 278 singletons and 32 pairs, while retaining all 342 instances. Each group is assigned wholly to one partition.

Each of the 23 categories with at least three instances receives a quota of one training and one validation instance. The seven most frequent eligible categories receive a second instance in each development partition, with frequency ties broken by label. Each partition therefore contains $16\times1+7\times2=30$ instances covering 23 categories. The official categories "Brush teeth" (two instances), "Get some water" (one), and "Take shower" (two) are assigned entirely to testing. These quotas prioritize category coverage rather than proportional sampling.

Following a seed-42 hash ordering of groups, we try assigning each group to Train, then Val, then Test. We commit the first choice for which an integer-programming feasibility check confirms that the remaining groups can still satisfy the remaining category quotas. All instances outside Train30 and Val30 form Test282, with no additional test subsampling. Table 5 summarizes the fixed partition.

Table 5: Frozen custom EAI split for VirtualHome Action Sequencing in `scene_1`. Category counts are distinct official activity labels within each partition.

Partition	Instances	Categories	Use
Train	30	23	Harness evolution
Val	30	23	Candidate selection
Test	282	26	Evaluation
Total	342	26	All `scene_1` instances

Train30 contains only singleton groups. We divide it deterministically into three disjoint Train10 batches, each containing ten distinct activity labels. Their union is Train30. These counts describe instance membership, not the total number of training rollouts.

B BENCHMARK HARNESS ARTIFACTS

B.1 VLABENCH

B.1.1 EVOLVED HARNESS

The following boxes reproduce the planner skill and context-construction code obtained after EnactOpt evolution for VLABench.

VLABench (EnactOpt): Planner Skill

```
# Primitive Planner Skill

You are the high-level planner for a **primitive** VLABench manipulation task.
Each episode is a single short manipulation, not a long multi-stage plan.

## What you see
Each round you receive:
- the task instruction
- the current camera views
- the prior execution history
- the scene spellings in `entities`
- the current state, including what is held and the end-effector pose

## Your job
Emit exactly **one** short command for the low-level controller.

The controller is frozen and weak at paraphrase. It works best on short,
    canonical
commands that preserve the benchmark's exact target and container tokens.

## Core rule
Use the shortest command that still preserves the task's exact target and final
desired state.

Do **not**:
- invent new object names
- normalize token-like labels into space-separated words
- add extra spatial qualifiers
- add strategy text or explanations
- repeat a failed command unchanged if the visible scene has changed

## Preserve exact scene spellings
When the scene provides a token-like label, copy that spelling exactly.
Keep underscores, capitals, and container suffixes exactly as shown in `
    entities`
and in the task instruction.

Examples of exact spellings:
- `contract_law_in_japan`
- `CuCl2`
- `jumpstart`
- `fridge_open`
- `giftbox_seen`
- `ace_of_hearts`

Do not rewrite these into natural language forms.

## Canonical command forms
Prefer these literal patterns when they match the task:

- `Add <condiment> to the dish`
- `Please take the book <title>`
- `Take out the <chemical> solution`
- `Take out the <drink> from the fridge_open`
- `Please pick the poker <card>`
```

VLABench (EnactOpt): Planner Skill (continued)

```
- `Put the <toy> into the giftbox_seen`

If the instruction already matches one of these forms, keep the command equally
short and literal, preserving the exact target token.

## Poker tasks
For poker episodes, the card name must be the exact scene entity spelling,
such as `ace_of_hearts` or `7_of_hearts`.
Do not write card names with spaces.

Example:
- task/instruction: `Please pick the poker ace of hearts`
- preferred subgoal: `Please pick the poker ace_of_hearts`

## History use
Look at the latest scene and history to decide whether the task is already done
    ,
partially done, or still untouched.

- If the target is untouched, issue the direct canonical command.
- If the target is already grasped or removed, keep the same final-state goal
    but
  do not add new narration.
- If a previous attempt failed and the visible scene changed, adapt only by
  choosing the still-missing primitive outcome.

Do not blindly repeat a failed sentence if the scene has moved on.

## Style
- one command only
- no bullets in the subgoal
- no commentary inside the command
- no mention of cameras, state, or history

## Good behavior
- stay literal
- preserve exact benchmark tokens
- use the smallest command that can still be executed
- avoid paraphrases that change the object spelling
```

VLABench (EnactOpt): Context Code

```
"""Context builder for primitive VLABench manipulation tasks.

This version keeps the context compact and grounded:
- preserve literal task wording
- surface current state facts (held objects, ee pose, entities, track)
- summarize history briefly instead of replaying everything verbatim
- attach only the most recent completed-round frames plus live views
- keep live forward/right/wrist images last so truncation preserves them
"""

from __future__ import annotations

CAMERA_ORDER = ("forward", "right", "wrist")
MAX_HISTORY_TEXT_ROUNDS = 4
MAX_HISTORY_IMAGE_ROUNDS = 2


def _map_get(obj, key, default=None):
    if obj is None:
        return default
    try:
        getter = obj.get
    except Exception:
        getter = None
    if getter is not None:
        try:
            return getter(key, default)
        except Exception:
            pass
    try:
        return obj[key]
    except Exception:
        return default


def _short(text, limit=120):
    if text is None:
        return ""
    try:
        s = str(text)
    except Exception:
        s = repr(text)
    s = s.replace("\n", " ").strip()
    if len(s) <= limit:
        return s
    return s[: max(0, limit - 1)].rstrip() + "..."


def _fmt_num(value):
    try:
        return f"{float(value):.3f}"
    except Exception:
        return _short(value, 20)


def _iter_items(value):
    if value is None:
        return []
    if isinstance(value, str):
        return [value]
    if hasattr(value, "tolist") and not isinstance(value, (list, tuple, set,
    dict)):
        try:
            value = value.tolist()
        except Exception:
            pass
    if isinstance(value, dict):
```

VLABench (EnactOpt): Context Code (continued)

```
        items = []
        for k, v in value.items():
            if isinstance(k, str):
                items.append(k)
            elif isinstance(v, str):
                items.append(v)
        if items:
            return items
        return [str(value)]
    try:
        return list(value)
    except Exception:
        return [value]


def _format_sequence(value, limit_items=8, item_limit=40):
    items = []
    seen = set()
    for item in _iter_items(value):
        s = _short(item, item_limit)
        if not s:
            continue
        if s in seen:
            continue
        seen.add(s)
        items.append(s)
        if len(items) >= limit_items:
            break
    if not items:
        return ""
    extra = 0
    try:
        total = len(list(_iter_items(value)))
        extra = max(0, total - len(items))
    except Exception:
        extra = 0
    text = ", ".join(items)
    if extra:
        text += f", ...(+{extra})"
    return text


def _format_pose(ee_pose):
    if ee_pose is None:
        return ""
    try:
        if hasattr(ee_pose, "tolist"):
            ee_pose = ee_pose.tolist()
    except Exception:
        pass
    if isinstance(ee_pose, (list, tuple)) and ee_pose:
        vals = list(ee_pose)
        if len(vals) >= 4:
            pos = ", ".join(_fmt_num(v) for v in vals[:3])
            grip = _fmt_num(vals[-1])
            return f"pos=[{pos}], gripper={grip}"
        return "[" + ", ".join(_short(v, 20) for v in vals) + "]"
    return _short(ee_pose, 80)


def _first_nonempty(*values):
    for v in values:
        if v is None:
            continue
        if isinstance(v, str) and not v.strip():
            continue
        if v == {} or v == [] or v == ():
```

VLABench (EnactOpt): Context Code (continued)

```
            continue
        return v
    return None


def _round_summary(index, record):
    planner = _map_get(record, "planner_output", {}) or {}
    parts = []

    sent = _first_nonempty(
        _map_get(record, "actor_subgoal", None),
        _map_get(planner, "subgoal", None),
    )
    if sent is not None:
        parts.append(f"sent={_short(sent, 120)!r}")

    steps_used = _first_nonempty(
        _map_get(record, "actor_steps_used", None),
        _map_get(record, "steps_used", None),
    )
    step_budget = _first_nonempty(
        _map_get(record, "actor_step_budget", None),
        _map_get(record, "step_budget", None),
    )
    if steps_used is not None:
        if step_budget is not None:
            parts.append(f"steps={steps_used}/{step_budget}")
        else:
            parts.append(f"steps={steps_used}")

    status = _first_nonempty(
        _map_get(record, "status", None),
        _map_get(record, "actor_status", None),
        _map_get(record, "execution_status", None),
    )
    runtime_error = _first_nonempty(
        _map_get(record, "runtime_error", None),
        _map_get(record, "error", None),
    )
    if status is not None:
        parts.append(f"status={_short(status, 40)}")
    if runtime_error is not None:
        parts.append(f"error={_short(runtime_error, 80)}")

    progress = _first_nonempty(_map_get(planner, "progress", None))
    if progress is not None:
        parts.append(f"progress={_short(progress, 120)!r}")

    if not parts:
        parts.append("no recorded details")
    return f"r{index}: " + "; ".join(parts)


def _get_camera_frame(images, camera):
    frame = _map_get(images, camera, None)
    if frame is not None:
        return frame
    try:
        return getattr(images, camera)
    except Exception:
        return None


def build_context(task: str, history: list[dict], observation, skill: str, llm=
    None) -> dict:
    del skill  # read-only planner prompt is already selected; keep builder
    deterministic.
```

VLABench (EnactOpt): Context Code (continued)

```
    del llm    # no evidence-based benefit for extra summarization here.

    history = history or []
    extra = getattr(observation, "extra", None) or {}
    state = getattr(observation, "state", None) or {}
    live_images = getattr(observation, "images", None) or {}

    task_text = (task or _map_get(observation, "instruction", "") or "").strip
    ()
    if not task_text:
        task_text = "No task text provided."

    track = _map_get(extra, "track", None)
    entities_raw = _map_get(extra, "entities", None)
    grasped_raw = _map_get(state, "grasped", None)
    ee_pose_raw = _map_get(state, "ee_pose", None)

    entities_text = _format_sequence(entities_raw, limit_items=12, item_limit
    =50)
    held_text = _format_sequence(grasped_raw, limit_items=8, item_limit=50)
    ee_pose_text = _format_pose(ee_pose_raw)

    # Build current-state text first.
    state_lines = []
    if track is not None:
        state_lines.append(f"track: {_short(track, 50)}")
    if held_text:
        state_lines.append(f"held: {held_text}")
    else:
        state_lines.append("held: none reported")
    if ee_pose_text:
        state_lines.append(f"ee_pose: {ee_pose_text}")
    if entities_text:
        state_lines.append(f"scene entities: {entities_text}")

    # Summarize history compactly.
    history_lines = []
    if not history:
        history_lines.append("No completed rounds yet.")
    else:
        show_from = max(0, len(history) - MAX_HISTORY_TEXT_ROUNDS)
        if show_from > 0:
            history_lines.append(f"{show_from} earlier completed rounds omitted
     from text to reduce stale context.")
        for idx in range(show_from, len(history)):
            rec = history[idx] or {}
            history_lines.append(_round_summary(idx, rec))

    # Attach images: most recent completed rounds first, then live views last.
    images = []
    tags = []

    frame_history = _map_get(extra, "frame_history", None) or []
    if not isinstance(frame_history, list):
        try:
            frame_history = list(frame_history)
        except Exception:
            frame_history = []

    paired_rounds = min(len(frame_history), len(history))
    start_round = max(0, paired_rounds - MAX_HISTORY_IMAGE_ROUNDS)
    for idx in range(start_round, paired_rounds):
        frame_dict = frame_history[idx] or {}
        for camera in CAMERA_ORDER:
            frame = _get_camera_frame(frame_dict, camera)
            if frame is None:
                continue
```

VLABench (EnactOpt): Context Code (continued)

```
            images.append(frame)
            tags.append(f"round{idx}_after_{camera}")

    live_present = []
    for camera in CAMERA_ORDER:
        frame = _get_camera_frame(live_images, camera)
        if frame is None:
            continue
        images.append(frame)
        tags.append(f"now_{camera}")
        live_present.append(camera)

    text_sections = [
        "# Task\n" + task_text,
        "# Current state\n" + "\n".join(state_lines),
        "# History\n" + "\n".join(history_lines),
        "# Views\n"
        + (
            "Attached images are ordered from older history to the live scene;
    the live forward/right/wrist views are last."
            if images
            else "No camera images available."
        ),
    ]

    hints = []
    if track is not None:
        hints.append(f"track={_short(track, 60)}")
    if held_text:
        hints.append(f"held={held_text}")
    else:
        hints.append("held=none")
    if entities_text:
        hints.append(f"entities={entities_text}")
    if not history:
        hints.append("fresh round")
    hints.append("use exact scene spellings")
    hints.append("live views are last")

    metadata = {
        "builder": "context_compact_state_v1",
        "task": task_text,
        "track": track,
        "history_rounds": len(history),
        "history_text_rounds_shown": len(history_lines) - (1 if len(history) >
    MAX_HISTORY_TEXT_ROUNDS else 0),
        "history_image_rounds_shown": paired_rounds - start_round,
        "images_attached": len(images),
        "current_cameras": live_present,
        "held": _iter_items(grasped_raw) if grasped_raw is not None else [],
        "ee_pose": ee_pose_text,
        "entities_preview": _iter_items(entities_raw)[:12] if entities_raw is
    not None else [],
    }

    return {
        "text": "\n\n".join(text_sections),
        "images": images,
        "image_tags": tags,
        "hints": hints,
        "metadata": metadata,
    }
```

B.2 ALFWORLD

B.2.1 EVOLVED HARNESS

The following boxes reproduce the final evolved ALFWorld skill and context code. The skill's original `H0` heading is retained verbatim.

ALFWorld (EnactOpt): Skill

```
# ALFWorld H0: SkillOpt-Aligned Task Strategy

## Interface

Solve household tasks using text observations and the admissible commands
provided for the current step. Choose a command from that list, preserving
its exact object names and numbers. Return <think>...</think> followed by
<action>one command</action>. The environment alone determines task success.

## Six Task Families

| Task | Ordered plan |
| --- | --- |
| Pick and place | Locate the object, take it, reach the destination, place it.
    |
| Pick two and place | Locate, take and deliver the first object; repeat for a
   distinct second object. |
| Examine in light | Find and hold the object, reach the specified lamp, use
   the lamp. |
| Clean and place | Find and take the object, clean it at the sink, deliver it
   to the destination. |
| Heat and place | Find and take the object, heat it at the microwave, deliver
   it to the destination. |
| Cool and place | Find and take the object, cool it at the fridge, deliver it
   to the destination. |

## Decision Priorities

1. Read the goal and identify the target, required state change, destination
   and number of objects. Work through locate, acquire, transform and deliver.
2. Search surfaces and containers systematically. Open a closed container
   before concluding that it has no target. Keep a short frontier of
   unsearched locations from the latest evidence; once several checks have
   been negative or the frontier is exhausted, move to a genuinely new
   plausible location rather than revisiting unchanged spots.
3. When the required object is visible and reachable and your hand is free,
   take it immediately instead of continuing to wander.
4. Finish any required cleaning, heating or cooling before final delivery.
   Check the feedback; issuing a command does not prove its success.
5. Once carrying the required object in the required state, go directly to
   the destination and place it using the command in the admissible list.
6. Track distinct objects delivered and the remaining count. Do not confuse
   one successful placement with completion of a two-object task.
7. Avoid action loops. Do not issue the same command more than twice in a
   row without changed evidence. If a command produces "Nothing happens."
   or otherwise leaves the state unchanged, treat it as a failed
   transition and replan immediately by checking preconditions or searching
   an unexplored location rather than repeatedly toggling or examining an
   object.
8. Do not invent commands or object identifiers. A listed action is executable,
   not necessarily useful; choose it according to the task and current evidence
    .

## Common Errors

Do not ignore a reachable target, skip a required transformation, repeatedly
search an unchanged container, or declare completion without observed evidence.
Inventory and location may change after an action; use the latest feedback.
```

ALFWorld (EnactOpt): Context Code

```
HISTORY_LENGTH = 50


def _clip(text, limit=180):
    text = "" if text is None else str(text)
    text = text.replace("\n", " ")
    if len(text) > limit:
        return text[:limit].rstrip() + "..."
    return text


def build_context(task, history, observation, skill):
    recent = history[-HISTORY_LENGTH:]
    start = len(history) - len(recent)
    lines = []
    for index, record in enumerate(recent, start + 1):
        lines.append(
            f"[Observation {index}: '{_clip(record.get('observation', ''))}', "
            f"Action {index}: '{_clip(record.get('action', ''))}']"
        )
    return {
        "history_text": "\n".join(lines),
        "retained_turns": list(range(start, len(history))),
    }
```

B.3 EAI

B.3.1 Evolved Harness

The following boxes reproduce the evolved EAI skill and context-construction code.

EAI (EnactOpt): Skill

```
Generate a nonempty ordered action plan from the current state using only
    supported action names and exact object IDs.
When feedback reports format_error or parse_error before any actions execute,
    repair the JSON envelope first; do not change the task logic until the
    submission is parseable.
If proximity to a target object is uncertain, WALK or FIND to the exact object
    before any direct interaction, and do not assume room membership implies
    range.
Treat facing, proximity, and possession as separate prerequisites: if LOOKAT/
    WATCH fails, TURNTO the same target first and then retry the observation
    before continuing.
When execution feedback reports a failure, use the updated state and specific
    failure cause to repair that unmet precondition instead of replaying the
    same blocked prefix; if that branch remains blocked, switch to another
    valid branch rather than repeating the blocker.
Prefer the shortest legal sequence that matches the visible affordances, and
    keep repeated action-goal actions in the required order.
Keep attempting while budget remains; do not stop after a review or failed
    correction, do not declare success or abandon the task, and let the
    environment's success signal end execution.
```

EAI (EnactOpt): Context Code

```
import json


def _render_value(value):
    if value is None:
        return 'null'
    if isinstance(value, str):
        text = value.strip()
        return text if text else '<empty>'
    try:
        return json.dumps(value, ensure_ascii=True, sort_keys=True, indent=2)
    except TypeError:
        return str(value)


def _section(title, value):
    return f'{title}:\n{_render_value(value)}'


def build_context(task, history, observation, skill):
    prompt = task['llm_prompt']
    if not isinstance(prompt, str) or not prompt.strip():
        raise ValueError('The complete official baseline prompt is required')
    if not isinstance(skill, str) or not skill.strip():
        raise ValueError('A non-empty Skill is required')

    latest_history = None
    if isinstance(history, list) and history:
        latest_history = history[-1]

    runtime_sections = [
        _section('Current observation (authoritative state)', observation),
    ]
    if latest_history is not None:
        runtime_sections.append(_section('Latest decision history item',
    latest_history))
    runtime_sections.append(_section('Full decision history', history))

    return {
        'messages': [
            {'role': 'user', 'content': 'Official task prompt (initial problem
    statement):\n' + prompt},
            {'role': 'user', 'content': 'Skill guidance:\n' + skill},
            {
                'role': 'user',
                'content': 'Runtime evidence (authoritative current state; only
     executed actions have effects; use the latest feedback and state, not the
    stale initial prompt):\n'
                + '\n\n'.join(runtime_sections),
            },
        ]
    }
```