

Self-Evolving Embodied Agents via Skill-Harness Evolution

Peidong Wang^{1,2,*} Zhiming Ma^{1,*} Ying Chang^{2,*} Xufang Luo^{2,†}
 Xiaocui Yang¹ Shi Feng^{1,‡} Yuqing Yang² Dongsheng Li²

¹Northeastern University

²Microsoft Research

Abstract

Embodied agents are increasingly built as systems around foundation models, where performance depends not only on model weights but also on the skills, context, action interfaces, and execution harness surrounding the model. While supervised fine-tuning and reinforcement learning can adapt agents to new environments, they require additional data, rewards, and training runs; meanwhile, many train-free code-centric approaches rely on programmable robot APIs that may be unavailable in fixed-interface settings. We propose SHAPER, a self-evolving framework for train-free embodied adaptation that keeps model parameters frozen and improves the non-parametric agent system by evolving reusable skills and a context-code harness through target-environment rollouts. In SHAPER, the same frozen model can serve as both planner and optimizer, refining its external skills and context code harness without parameter updates. We evaluate SHAPER on VLABench and ESI-Bench, covering embodied agents with different low-level action interfaces, and compare against pure execution, supervised fine-tuning, and test-time-scaling baselines such as verifier-free selection and voting. Our results suggest that skill-and-harness optimization is a practical route to self-evolving embodied agents when model training is expensive, unavailable, or undesirable.

1 Introduction

Embodied AI is increasingly moving from isolated policy models toward agent systems. Recent vision-language-action models and multimodal foundation models have shown that large pretrained models can perceive scenes, interpret instructions, reason over objects and spatial relations, and act in simulated or physical environments Kim et al. [2025], Black et al. [2024, 2025], Physical Intelligence et al. [2026], NVIDIA et al. [2025], Gemini Robotics Team et al. [2025], Yuan et al. [2026]. At the same time, work on agent scaffolds, agent-computer interfaces, harness optimization, and robot skill discovery shows that an agent’s behavior is not determined by the model alone Yang et al. [2024], Chen et al. [2026], Pan et al. [2026a], Lee et al. [2026], Lou et al. [2026], Ju et al. [2026], Lu et al. [2026]. A deployed embodied agent is shaped by model-external components, including reusable skills, context construction, action interfaces, output parsing, and execution wrappers. These non-parametric choices are especially consequential in embodied settings, where an agent must act through constrained action interfaces under partial visual feedback.

A common approach to adapt embodied agents is still to update model parameters. Supervised Fine-Tuning (SFT) and Reinforcement-Learning (RL) post-training can improve embodied perfor-

*Equal contribution.

†Corresponding author: luoxufang@outlook.com.

‡Corresponding author: fengshi@cse.neu.edu.cn.

§Work done while Peidong Wang and Ying Chang were interns at Microsoft Research Asia.

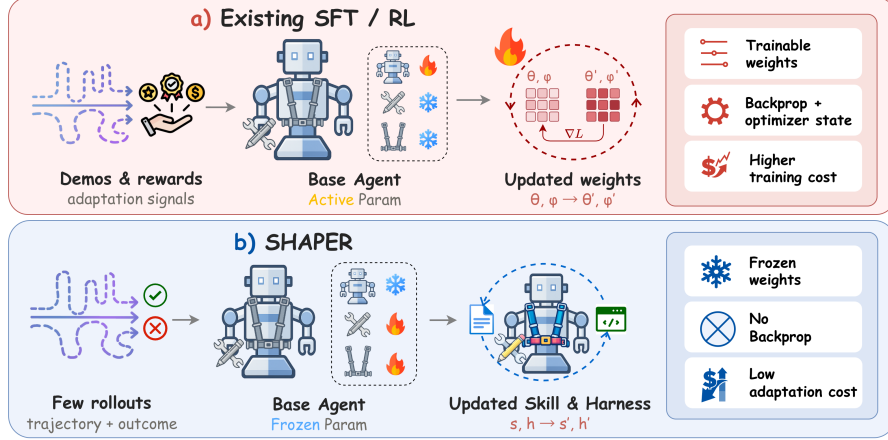


Figure 1: SHAPER adapts frozen embodied agents by evolving skills and harnesses from a few rollouts.

mance Open X-Embodiment Collaboration et al. [2023], Khazatsky et al. [2024], Guo et al. [2025], Lu et al. [2025], Pan et al. [2026b], Physical Intelligence et al. [2025], but require access to model weights, task-specific demonstrations or rewards, and additional optimization. These requirements limit adaptation when model weights are unavailable or interaction data are scarce. This motivates a complementary question: how can a strong frozen embodied agent be adapted to a target environment without parameter updates?

Train-free embodied adaptation offers a different path. Recent agentic robotics systems show that procedural knowledge can live outside the model: language models can write robot programs, revise failed executions, and accumulate reusable skills without directly changing neural weights Ahn et al. [2022], Liang et al. [2023], Huang et al. [2023], Wang et al. [2023], Ju et al. [2026], Lu et al. [2026]. However, much of this progress treats executable code as the action substrate. The agent improves by generating or repairing programs that call robot-specific APIs Liang et al. [2023], Lu et al. [2026], which is powerful when such APIs, debugging hooks, and execution monitors are available. In many embodied environments, these interfaces may be absent, incomplete, or difficult to expose, and the agent may instead have to act through a fixed or restricted action space. This leaves open how to systematically adapt embodied agents by improving the planner-side procedural guidance and context that shape behavior, rather than by synthesizing low-level control programs.

As illustrated in Figure 1, we propose SHAPER, a self-evolving framework for train-free embodied adaptation. The key distinction is what adaptation changes. SFT and RL turn demonstrations or rewards into updates to model weights through backpropagation; SHAPER instead turns a small set of target-environment trajectories and outcomes into revisions of reusable *skills* and a context-code *harness*, while keeping the planner and executor weights frozen. Here, a skill is a piece of procedural guidance that may be written as natural language, an action recipe, a failure-recovery rule, or a task-decomposition strategy. The context-code harness determines which observations, prior actions, execution outcomes, and feedback are presented to the planner and how they are organized. We reuse the same frozen VLM checkpoint in two distinct roles. During environment interaction, it acts as the upper-level planner. During evolution, it is prompted separately as an artifact optimizer that revises the textual skill and context-code harness from summarized rollout feedback. The two roles share model weights but use different prompts and inputs. Thus, the agent evolves its external skills and context construction while keeping its model parameters fixed.

We evaluate SHAPER across embodied environments with different low-level action interfaces. In VLABench Zhang et al. [2024], the upper-level VLM agent is connected to the environment through a VLA actor that serves as a low-level execution tool. In ESI-Bench Hong et al. [2026], the same agentic formulation acts through the benchmark’s environment action interface. Across these settings, we compare against pure execution, same-data SFT, and Test-Time-Scaling (TTS) baselines such as verifier-free selection and trajectory voting Jang et al. [2025], Lin et al. [2025]. Our results aim to show that evolving skills and harnesses is a practical way to improve embodied agents when parameter updates are expensive, unavailable, or undesirable.

This paper makes three contributions: (1) we formulate train-free embodied adaptation as non-parametric optimization of an embodied agent system, where reusable skills and context-code harnesses are optimized around a frozen model; (2) we introduce SHAPER, a self-evolving framework that improves an embodied agent by evolving textual skills and a context-code harness through target-environment rollouts, without updating model parameters; and (3) we evaluate SHAPER across embodied environments with different action interfaces, showing that skill-and-harness optimization can improve embodied performance and provide a competitive alternative to fine-tuning and sampling-heavy baselines.

2 Related Work

Embodied foundation models and parameter-updating adaptation. Recent embodied foundation models connect large-scale visual and language pretraining to robot control. RT-2, OpenVLA, Octo, and π_0 establish generalist VLA policies, while $\pi_{0.5}$, $\pi_{0.7}$, GR00T N1, Gemini Robotics, and Qwen-RobotManip extend open-world generalization, steerability, humanoid control, and embodied reasoning Zitkovich et al. [2023], Kim et al. [2025], Octo Model Team et al. [2024], Black et al. [2024, 2025], Physical Intelligence et al. [2026], NVIDIA et al. [2025], Gemini Robotics Team et al. [2025], Yuan et al. [2026]. Large robot datasets and parameter-updating adaptation through supervised or reinforcement learning further broaden their capabilities Open X-Embodiment Collaboration et al. [2023], Khazatsky et al. [2024], Guo et al. [2025], Lu et al. [2025], Pan et al. [2026b], Shi et al. [2026], Physical Intelligence et al. [2025]. These approaches remain dependent on additional data, rewards or corrections, and model updates; we instead adapt frozen agents through external skill and harness artifacts.

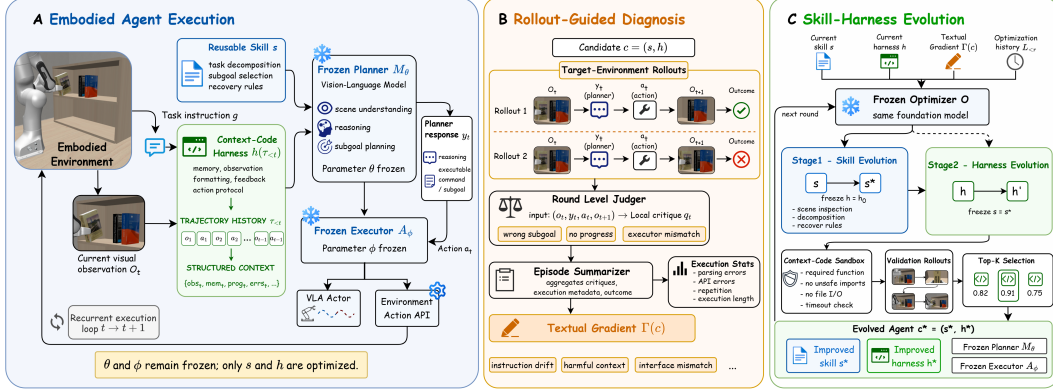
Self-evolving agents. Recent work adapts frozen agents by editing model-external artifacts. SkillOpt optimizes a persistent natural-language skill through bounded edits and validation-gated rollouts while keeping the execution harness fixed Yang et al. [2026]. EmbodiSkill brings skill evolution to embodied environments by distinguishing defective skill content from execution lapses, but likewise updates the skill rather than the context harness Ju et al. [2026]. AutoHarness instead synthesizes a code harness, or even a complete code policy, from environment feedback in text-game environments, without co-optimizing reusable procedural guidance Lou et al. [2026]. AgentSpec represents embodied scaffolds as typed, swappable compositions for controlled analysis rather than rollout-driven artifact optimization Chen et al. [2026]. In contrast, SHAPER evolves both the reusable skill and context-code harness of an embodied agent from environment interaction, while keeping all model parameters frozen.

Train-free adaptation for embodied agents. Train-free embodied systems compose frozen models with affordances, executable robot APIs, spatial value maps, or reusable skills Ahn et al. [2022], Liang et al. [2023], Huang et al. [2023], Ju et al. [2026], Lu et al. [2026]. Code- and API-centric approaches, however, depend on the programmable interfaces, perception modules, and execution monitors available on each platform. For VLA policies, vision-language steering, latent prompt optimization, recurrent-depth reasoning, verifier guidance, and test-time sampling or voting improve action execution without retraining Liu et al. [2026], Zhang et al. [2026], Tur et al. [2026], Singhi et al. [2026], Kwok et al. [2025], Jang et al. [2025], Lin et al. [2025], Yang et al. [2025]. SHAPER instead targets fixed action interfaces and uses rollout feedback to jointly evolve reusable high-level skills and harness context.

3 Method

3.1 Train-Free Agent Factorization

Modern embodied agents are rarely a single neural policy used in isolation. As shown in Figure 2A, we factor an embodied agent into four system components: a VLM planner M_θ , an executor $\mathcal{A}_\phi$, a reusable skill s , and a context-code harness h . The executor may be a VLA actor or a wrapper around an environment action API. The skill is textual procedural guidance in the planner prompt, covering scene inspection, task decomposition, subgoal selection, recovery, and termination. The harness is a context builder—implemented as a Python function such as `build_context`—that selects and



statistics form the textual gradient:

$$\begin{aligned} e^c(g) &= \text{Summarize}(m^c(g), h(\tau_{<T}), \{q_t^c(g)\}_{t=0}^{T-1}, \\ &\quad R(\tau^c(g))), \\ \Gamma(c) &= \text{Concat}(\{e^c(g) : g \in \mathcal{D}_{\text{tr}}^r\}, \text{Stats}(c)). \end{aligned} \quad (3)$$

The resulting $\Gamma(c)$ exposes actionable cross-round patterns such as unproductive repetition, instruction drift, missing progress tracking, harmful context, and parsing or API errors.

3.3 Two-Stage Skill-Harness Evolution

Starting from $c_0 = (s_0, h_0)$, SHAPER maintains a beam $\mathcal{B}_r$ of candidates. As shown in Figure 2C, the same frozen foundation model used by the planner is invoked under a separate optimizer prompt. Conditioned on the current candidate, its textual gradient, and optimization history $\mathcal{L}_{<r}$, the optimizer proposes

$$c' = (s', h') \sim \mathcal{O}(c, \Gamma(c), \mathcal{L}_{<r}), \quad (4)$$

where $\mathcal{O}$ denotes this optimizer role. We use a two-stage schedule: skill evolution first updates s with the seed harness fixed, after which harness evolution updates h with the selected skill fixed:

$$\begin{aligned} s' &\sim \mathcal{O}_s(s, h_0, \Gamma, \mathcal{L}), \quad h' = h_0, \\ h' &\sim \mathcal{O}_h(s^*, h, \Gamma, \mathcal{L}), \quad s' = s^*. \end{aligned} \quad (5)$$

Generated harnesses are sandboxed before rollout: they must define the required context function, avoid imports, file I/O, and dynamic execution, and finish within a timeout. Invalid candidates are rejected and their validation errors are returned as optimizer feedback. Valid candidates are evaluated on a fixed validation set $\mathcal{D}_{\text{val}}$, and the beam retains the top- K incumbents and proposals:

$$\begin{aligned} \hat{J}_{\text{val}}(c) &= \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{g \in \mathcal{D}_{\text{val}}} R(\tau^c(g)), \\ \mathcal{B}_{r+1} &= \text{TopK}(\mathcal{B}_r \cup \mathcal{C}_r; \hat{J}_{\text{val}}, K), \end{aligned} \quad (6)$$

where $\mathcal{C}_r$ is the set of newly proposed candidates. The final output is the highest-scoring candidate $c^* = (s^*, h^*)$ encountered across rounds. It retains the frozen planner and executor while changing only the planner’s textual guidance and constructed context.

4 Experiments

We evaluate whether skill-harness evolution improves a frozen upper-level VLM planner across two embodied environments with different execution interfaces. In VLABench, the planner delegates textual subgoals to a frozen VLA actor; in ESI-Bench, it interacts through the benchmark’s fixed action API. Across both environments, SHAPER keeps all planner and executor parameters fixed and changes only the external skill and harness artifacts.

4.1 Experimental Setup

Benchmarks. VLABench Zhang et al. [2024] is a language-conditioned manipulation benchmark targeting implicit language understanding, common-sense and world-knowledge transfer, and long-horizon reasoning, all of which are underrepresented in the earlier manipulation benchmarks considered. Its instructions often imply rather than explicitly state goals, and task instances vary object categories to test generalization.

Table 1: VLABench split design.

Task form	Target category	
	Seen	Unseen
Semantic understanding	C1	C2
Common sense	C3	C4

We focus on Semantic Understanding, which tests recovery of intended goals from nuanced language, and Common Sense & World Knowledge, which requires grounding prior knowledge in the observed scene.

Method	C1	C2	C3	C4	Overall
VLA-level baselines					
	no upper-level planner				
Direct VLA	26.0	9.5	45.0	12.5	23.25
SFT (same data)	32.5	13.0	38.0	12.5	24.00
Test-time scaling					
	inference-time aggregation				
VLA + MG-Select	25.0	12.0	29.5	19.0	21.38
VLA + VOTE	26.0	12.5	33.0	16.5	22.00
Seed Agent + MG-Select	35.0	15.0	40.5	7.0	24.38
Seed Agent + VOTE	38.0	13.0	34.0	9.0	23.50
Train-free agent adaptation					
	frozen planner and executor				
Seed Agent	40.0	20.0	40.0	13.0	28.25
Skill Evolution	42.0	23.5	50.0	18.5	33.50
Harness Evolution	42.0	21.0	46.0	13.0	30.50
SHAPER	42.5	26.0	50.0	19.5	34.50

Table 2: VLABench success rates (%) on four 200-episode splits; best results are bolded.

As summarized in Table 1, C1–C4 are our custom held-out evaluation splits rather than official VLABench splits. C1 is fully in-domain; C2 uses unseen target categories; C3 changes the task form to Common Sense & World Knowledge; and C4 shifts both. Each cell contains five task families and 40 held-out episodes per family, giving 200 episodes per cell and 800 episodes overall.

ESI-Bench Hong et al. [2026] evaluates embodied spatial intelligence in a closed perception–action loop spanning 10 task categories. Unlike passive spatial question answering, agents must actively acquire evidence unavailable from the initial view through perception, locomotion, and manipulation. We evaluate on a 231-question subset sampled to match the official benchmark’s category proportions and report both overall and category-level accuracy.

Models and execution interfaces. All agent-system variants use Qwen3.6-27B as the frozen upper-level planner at evaluation. On VLABench, the executor is the official VLABench π_0 checkpoint, fine-tuned on the benchmark’s ten primitive task categories and served through OpenPI. It receives a textual subgoal and produces low-level controls in five-action chunks. The upper-level planner specifies an environment-step budget for each subgoal, so one planner round may execute multiple chunks. An episode permits at most 10 planner rounds and 400 low-level environment steps. On ESI-Bench, the same planner model acts through the benchmark’s fixed interaction API for at most 30 steps.

Optimization protocol. For VLABench, we evolve the artifacts on 15 training episodes and select candidates on a fixed 24-episode validation set; both are disjoint from the 800 held-out evaluation episodes. For ESI-Bench, we use 10 training questions and 10 validation questions, all disjoint from the 231-question evaluation subset. For both benchmarks, evolution uses four beam-search rounds with width 3 and branch factor 2, and textual feedback is formed from minibatches of 4 rollouts. We first optimize the skill and then optimize the harness while holding the evolved skill fixed.

Compared methods. We organize VLABench baselines by what they modify. *Executor-level baselines* include direct π_0 execution and SFT of the same actor architecture on the same 15 training episodes. *Test-time scaling* includes MG-Select Jang et al. [2025] and VOTE Lin et al. [2025] applied either directly to the VLA actor or on top of the seed planner. *Agent-system adaptation* includes Seed Agent, Skill Evolution, Harness Evolution, and full Skill-Harness Evolution. On ESI-Bench, we compare the seed, evolved-skill, and full variants and show GPT-5 Passive Single-view (PS) from the benchmark paper only as an external published reference.

4.2 VLABench Results

Main results. Table 2 shows that the Seed Agent reaches 28.25%, improving direct VLA execution by 5.00 points and same-data SFT by 4.25 points without changing either model. Full skill-harness evolution further raises success to 34.50%, 6.25 points above the seed and 10.50 points above

Category	n	Seed Agent	Evolved Skill	SHAPER	<i>GPT-5 PS</i> [†]
Physical Structure	21	14.3	38.1	38.1	51.4
Physical Dynamics	10	40.0	40.0	40.0	46.1
Specular Reflection	20	10.0	20.0	60.0	53.0
Perceptual Grounding	49	53.1	59.2	69.4	24.9
Metric Comparison	21	33.3	47.6	47.6	48.2
Enumerative Perception	18	16.7	33.3	27.8	10.7
Spatial Relations	51	27.5	37.3	54.9	31.0
Cognitive Mapping	17	29.4	23.5	23.5	60.4
Temporal Understanding	19	47.4	47.4	47.4	40.6
Action Sequencing	5	40.0	40.0	20.0	36.4
Micro Avg.	231	32.5	41.1	49.8	–
Macro Avg.	10 categories	31.2	38.6	42.9	40.3

Table 3: ESI-Bench accuracy (%) on the official-proportion 231-question subset. n denotes the number of evaluated questions. *GPT-5 PS*[†] is the published Passive Single-view result on the full evaluation set.

SFT. In contrast, VLA-level MG-Select and VOTE reach 21.38% and 22.00%, both below direct VLA execution, while their seed-agent counterparts reach 24.38% and 23.50%, below the unscaled Seed Agent. Thus, the improvements from artifact evolution are not reproduced by additional inference-time sampling or aggregation in this setting.

Generalization across shifts. The four splits separately vary target category and task form. Full SHAPER reaches 42.5% on the in-domain C1 split, compared with 40.0% for the seed. Its descriptive gains become larger under distribution shift: +6.0 points for unseen targets (C2), +10.0 for an unseen task form with seen targets (C3), and +6.5 when both are unseen (C4). It obtains the best result on C1, C2, and C4 and ties Skill Evolution at 50.0% on C3. The largest gain therefore occurs when transferring the evolved artifacts from Semantic Understanding to the Common Sense & World Knowledge task form.

Artifact ablations. Skill-only, harness-only, and full evolution score 33.50%, 30.50%, and 34.50%, respectively, versus 28.25% for the seed. Skill evolution accounts for the largest descriptive difference from the seed (+5.25 points), while harness-only is +2.25 points. Adding the evolved harness after skill evolution changes the overall score by +1.00 point; the split-wise differences are +0.5 on C1, +2.5 on C2, 0.0 on C3, and +1.0 on C4. The effects are not additive: for example, harness-only raises C3 from 40.0% to 46.0%, but the full and skill-only configurations both obtain 50.0%. We therefore interpret these rows as complete agent configurations rather than estimates of independent artifact effects.

Actor-aware recovery. Figure 3 compares Seed Agent and Skill Evolution on `select_book/ep_024` with identical context construction. The seed skill asks the planner to inspect the task, camera observations, and execution history before producing a subtask. It does not specify the VLA actor’s preferred command distribution, distinguish selection from retrieval or placement, enforce exact entity names, or require an explicit progress check. Consequently, after the target book is partly pulled from the shelf, the planner treats continued visibility as evidence that the same retrieval command should be repeated. The episode reaches the 400-step limit after three planner rounds with reward 0.5.

The evolved skill turns the planner into an actor-aware command adapter. It routes task intents to short canonical commands, emits one primitive per round, preserves the complete book title, and excludes low-level details such as grasp orientation or approach geometry handled by the VLA. It requires the planner to classify the previous attempt as successful, failed, partially successful, or stalled, and to change the canonical command when repeated execution makes no progress. In the same episode, the planner detects remaining shelf overlap, switches from the original retrieval command to a pick-up command, and later recognizes that the book is clear of the shelf. It obtains reward 1.0 in the same three planner rounds using 297 environment steps.

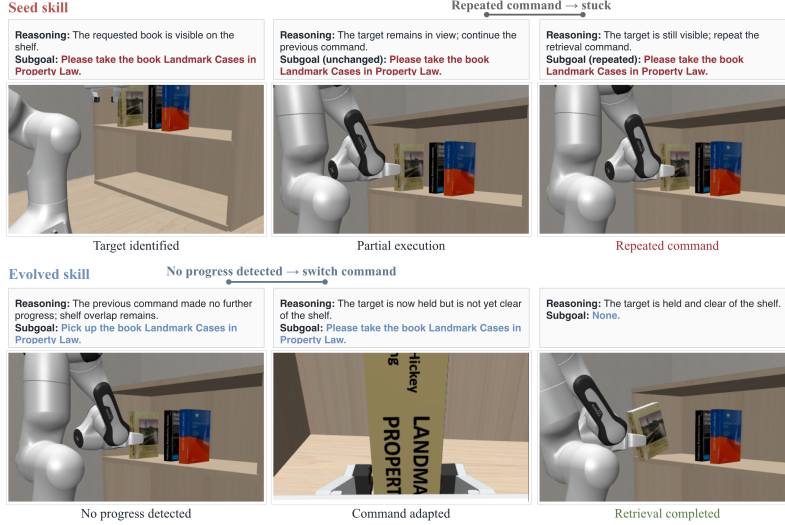


Figure 3: Controlled VLABench case isolating skill evolution for the instruction “Fetch the Landmark Cases in Property Law book for tomorrow’s class.” The planner, VLA executor, and raw harness are fixed. The seed skill repeats a retrieval command after partial execution, whereas the evolved skill detects no progress, changes the command, and completes retrieval. Text annotations summarize observable decisions rather than quote hidden reasoning.

4.3 ESI-Bench Results

Main results. Table 3 shows that full SHAPER reaches 49.8% micro and 42.9% macro accuracy, versus 32.5%/31.2% for the Seed Agent and 41.1%/38.6% for Skill Evolution. Full evolution is therefore 17.3 micro points and 11.7 macro points above the seed, and 8.7 micro points and 4.3 macro points above skill-only. Its macro score is numerically above the published GPT-5 PS reference (42.9% vs. 40.3%), placing the frozen 27B planner in a similar aggregate range without parameter updates. This remains an external rather than paired comparison: GPT-5 PS uses the full Passive Single-view setting, whereas our methods use the fixed 231-question subset.

Category-level behavior. Skill Evolution improves six of the ten categories over the seed, including Physical Structure, Specular Reflection, Perceptual Grounding, Metric Comparison, Enumerative Perception, and Spatial Relations. Adding the evolved harness produces its clearest positive differences on Specular Reflection (20.0% to 60.0%), Perceptual Grounding (59.2% to 69.4%), and Spatial Relations (37.3% to 54.9%). Five categories remain unchanged, while Enumerative Perception decreases from 33.3% to 27.8% and Action Sequencing from 40.0% to 20.0%. These results are descriptive and should be interpreted with their sample counts; Action Sequencing, for example, contains only five questions. The following reflection case illustrates the context behavior behind one of the largest positive differences.

Evidence-aware visual memory. Figure 4 compares Skill Evolution with full SHAPER under the same evolved skill. The raw harness concatenates the complete textual action and reasoning history while retaining only the five most recent RGB observations. This recency-based window gives repeated or blank views the same status as earlier task-critical evidence, and the uncompressed text may preserve the model’s own mistaken observations. In the shown episode, the agent sees one target jar in the mirror at step 1, but that visual anchor is no longer available when the three real candidates are later observed. It spends all 30 steps oscillating between views, taking 15 right turns, 12 left turns, and 3 forward moves before answering “not sure.”

The evolved harness replaces this window with sparse evidence-aware memory. It selects up to three informative and visually diverse keyframes from the full history, pairs each RGB observation with a deterministic focus crop, and compresses the action path and recent execution record. For the current view, it additionally supplies selected horizontal-band and overlapping-tile crops. All crops are computed from official RGB pixels using contrast, edge density, and sharpness, without target boxes

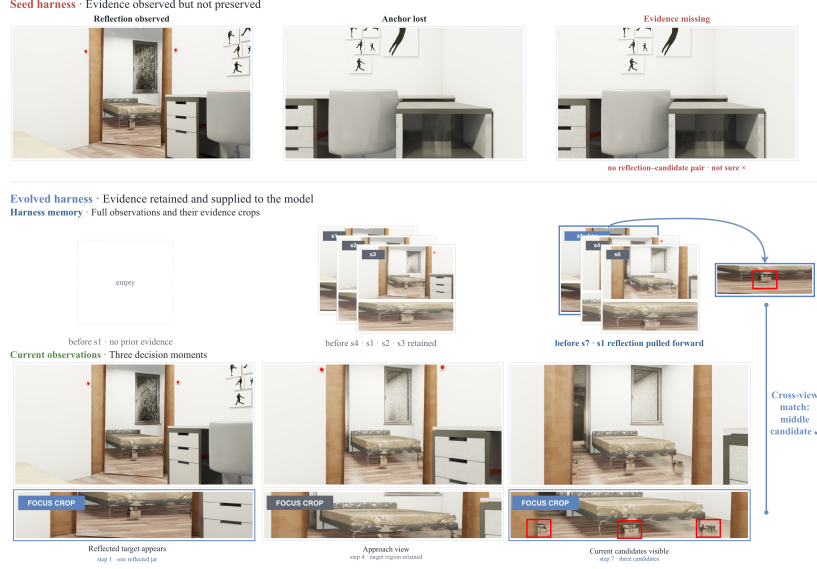


Figure 4: ESI-Bench case isolating harness evolution for the instruction “You are given an image with a mirror and three noodle_jars. Based on the mirror reflection, which of the three noodle_jars in the same image corresponds to the reflected one: the left, middle, or right object?” The planner, action interface, and evolved skill are fixed. The raw harness drops the early reflection evidence, whereas the evolved harness retains the reflected target and a deterministic focus crop for cross-view matching, correctly selecting the middle candidate.

or simulator metadata. For reflection tasks, the constructed context explicitly preserves one frame of the reflected target and another of the real candidates for appearance matching. Before acting, the planner reports structured VIEW, TARGETS, EVIDENCE, READY, and GAP fields, while the harness flags low-information repetition and inverse-action cycles. In Figure 4, the step-1 reflection remains available at step 7 for comparison with the three candidates. After six forward moves, the agent answers “middle” with confidence 0.90. All visual evidence comes from official RGB observations and reference images; the harness never accesses pose, depth, or other metadata.

4.4 Optimization Efficiency

Acquiring the evolved artifacts is a one-time cost. The logged token usage corresponds to approximately \$2.25 for VLABench and \$2.83 for ESI-Bench in API-equivalent cost per evolution run. These estimates include planner rollouts, judging, episode summarization, and artifact optimization, but exclude final evaluation and simulator or GPU infrastructure. After evolution, the selected skill and harness are reused across all held-out episodes without per-episode search; unlike test-time scaling, their acquisition cost does not grow with the number of deployments.

5 Conclusion

We presented SHAPER, a train-free approach that improves embodied agents by evolving a textual skill and context-code harness while keeping planner and executor parameters fixed. Rollout-derived textual feedback drives a two-stage optimization that produces reusable procedural guidance and trajectory context rather than per-episode action searches. Across VLABench and ESI-Bench, the same frozen 27B upper-level planner improves with both a VLA executor and an active-perception action interface, including under held-out target categories and task forms. These results support skill-harness evolution as a lightweight approach to adapting frozen embodied agents, while cross-embodiment transfer and real-robot validation remain for future work.

References

- Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pages 2679–2713. PMLR, 2025. URL <https://proceedings.mlr.press/v270/kim25c.html>.
- Kevin Black et al. π_0 : A vision-language-action flow model for general robot control, 2024. URL <https://arxiv.org/abs/2410.24164>.
- Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, brian ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 17–40. PMLR, 2025. URL <https://proceedings.mlr.press/v305/black25a.html>.
- Physical Intelligence, Bo Ai, et al. $\pi_{0.7}$: a steerable generalist robotic foundation model with emergent capabilities, 2026. URL <https://arxiv.org/abs/2604.15483>.
- NVIDIA, Johan Bjorck, et al. GR00T N1: An open foundation model for generalist humanoid robots, 2025. URL <https://arxiv.org/abs/2503.14734>.
- Gemini Robotics Team, Saminda Abeyruwan, et al. Gemini robotics: Bringing ai into the physical world, 2025. URL <https://arxiv.org/abs/2503.20020>.
- Haoqi Yuan, Zhixuan Liang, Anzhe Chen, Ye Wang, Haoyang Li, Pei Lin, Yiyang Huang, Zixing Lei, Tong Zhang, Jiazhao Zhang, Jie Zhang, Jingyang Fan, Gengze Zhou, Qihang Peng, Chenxu Lv, Xiaoyue Chen, An Yang, Fei Huang, Junyang Lin, Dayiheng Liu, Jingren Zhou, Chenfei Wu, and Xiong-Hui Chen. Qwen-RobotManip technical report: Alignment unlocks scale for robotic manipulation foundation models, 2026. URL <https://arxiv.org/abs/2606.17846>.
- John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering, 2024. URL <https://arxiv.org/abs/2405.15793>.
- Jixuan Chen et al. AgentSpec: Understanding embodied agent scaffolds through controlled composition, 2026. URL <https://arxiv.org/abs/2606.14674>.
- Linyue Pan, Lexiao Zou, Shuo Guo, Jingchen Ni, and Hai-Tao Zheng. Natural-language agent harnesses, 2026a. URL <https://arxiv.org/abs/2603.25723>.
- 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>.
- Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P. Murphy. Autoharness: improving llm agents by automatically synthesizing a code harness, 2026. URL <https://arxiv.org/abs/2603.03329>.
- 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/2605.10332>.
- 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 Fan, and Guanzhi Wang. ASPIRE: Agentic /skills discovery for robotics, 2026. URL <https://arxiv.org/abs/2607.00272>.

- Open X-Embodiment Collaboration, Abby O’Neill, et al. Open x-embodiment: Robotic learning datasets and RT-X models, 2023. URL <https://arxiv.org/abs/2310.08864>.
- Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. DROID: A large-scale in-the-wild robot manipulation dataset, 2024. URL <https://arxiv.org/abs/2403.12945>.
- Yanjiang Guo, Jianke Zhang, Xiaoyu Chen, Xiang Ji, Yen-Jen Wang, Yucheng Hu, and Jianyu Chen. Improving vision-language-action model with online reinforcement learning, 2025. URL <https://arxiv.org/abs/2501.16664>.
- Guanxing Lu, Wenkai Guo, Chubin Zhang, Yuheng Zhou, Haonan Jiang, Zifeng Gao, Yansong Tang, and Ziwei Wang. VLA-RL: Towards masterful and general robotic manipulation with scalable reinforcement learning, 2025. URL <https://arxiv.org/abs/2505.18719>.
- Mingjie Pan, Siyuan Feng, Qinglin Zhang, Xinchun Li, Jianheng Song, Chendi Qu, Yi Wang, Chuankang Li, Ziyu Xiong, Zhi Chen, Yi Liu, and Jianlan Luo. SOP: A scalable online post-training system for vision-language-action models, 2026b. URL <https://arxiv.org/abs/2601.03044>.
- Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, et al. $\pi_{0.6}^*$: a VLA that learns from experience, 2025. URL <https://arxiv.org/abs/2511.14759>. Blog: <https://www.pi.website/blog/pistar06>.
- Michael Ahn et al. Do as i can, not as i say: Grounding language in robotic affordances, 2022. URL <https://arxiv.org/abs/2204.01691>.
- 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)*, pages 9493–9500. IEEE, 2023. doi: 10.1109/ICRA48891.2023.10160591. URL <https://ieeexplore.ieee.org/document/10160591>.
- Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 540–562. PMLR, 2023. URL <https://proceedings.mlr.press/v229/huang23b.html>.
- 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, 2023. URL <https://arxiv.org/abs/2305.16291>.
- Shiduo Zhang, Zhe Xu, Peiju Liu, Xiaopeng Yu, Yuan Li, Qinghui Gao, Zhaoye Fei, Zhangyue Yin, Zuxuan Wu, Yu-Gang Jiang, and Xipeng Qiu. VLABench: A large-scale benchmark for language-conditioned robotics manipulation with long-horizon reasoning tasks, 2024. URL <https://arxiv.org/abs/2412.18194>.
- Yining Hong, Jiageng Liu, Han Yin, Manling Li, Leonidas Guibas, Li Fei-Fei, Jiajun Wu, and Yejin Choi. ESI-Bench: Towards embodied spatial intelligence that closes the perception-action loop, 2026. URL <https://arxiv.org/abs/2605.18746>.
- Suhyeok Jang, Dongyoung Kim, Changyeon Kim, Youngsuk Kim, and Jinwoo Shin. Verifier-free test-time sampling for vision language action models, 2025. URL <https://arxiv.org/abs/2510.05681>.
- Juyi Lin, Amir Taherin, Arash Akbari, Arman Akbari, Lei Lu, Guangyu Chen, Taskin Padi, Xiaomeng Yang, Weiwei Chen, Yiqian Li, Xue Lin, David Kaeli, Pu Zhao, and Yanzhi Wang. VOTE: Vision-language-action optimization with trajectory ensemble voting, 2025. URL <https://arxiv.org/abs/2507.05116>.

- Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2165–2183. PMLR, 2023. URL <https://proceedings.mlr.press/v229/zitkovich23a.html>.
- Octo Model Team, Dibya Ghosh, et al. Octo: An open-source generalist robot policy, 2024. URL <https://arxiv.org/abs/2405.12213>.
- Liangzhi Shi, Shuaihang Chen, Feng Gao, Yinuo Chen, Kang Chen, Tonghe Zhang, Hongzhi Zang, Jiakai Zhou, Weinan Zhang, Chao Yu, and Yu Wang. Beyond imitation: Reinforcement learning-based sim-real co-training for VLA models, 2026. URL <https://arxiv.org/abs/2602.12628>.
- 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/2605.23904>.
- Shuo Liu, Ishneet Sukhvinder Singh, Yiqing Xu, Jiafei Duan, and Ranjay Krishna. VLS: Steering pretrained robot policies via vision-language models, 2026. URL <https://arxiv.org/abs/2602.03973>.
- Wenbo Zhang, Jianxiong Li, Shuai Yang, Sijin Chen, Jiajun Liu, Lingqiao Liu, and Xiao Ma. TTT-VLA: Test-time latent prompt optimization for vision-language-action models, 2026. URL <https://arxiv.org/abs/2606.03127>.
- Yalcin Tur, Jalal Naghiyev, Haoquan Fang, Wei-Chuan Tsai, Jiafei Duan, Dieter Fox, and Ranjay Krishna. Recurrent-depth VLA: Implicit test-time compute scaling of vision-language-action models via latent iterative reasoning, 2026. URL <https://arxiv.org/abs/2602.07845>.
- Nishad Singhi, Christian Bialas, Snehal Jauhri, Vignesh Prasad, Georgia Chalvatzaki, Marcus Rohrbach, and Anna Rohrbach. Think twice, act once: Verifier-guided action selection for embodied agents, 2026. URL <https://arxiv.org/abs/2605.12620>.
- Jacky Kwok, Christopher Agia, Rohan Sinha, Matt Foutter, Shulu Li, Ion Stoica, Azalia Mirhoseini, and Marco Pavone. Robomongo: Scaling test-time sampling and verification for vision-language-action models. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 3200–3217. PMLR, 2025. URL <https://proceedings.mlr.press/v305/kwok25a.html>.
- Siyuan Yang, Yang Zhang, Haoran He, Ling Pan, Xiu Li, Chenjia Bai, and Xuelong Li. Steering vision-language-action models as anti-exploration: A test-time scaling approach, 2025. URL <https://arxiv.org/abs/2512.02834>.

A Additional Artifact and Optimization Details

This appendix makes the optimization targets and feedback pipeline concrete by reproducing the artifacts and prompt templates used in our experiments. A *skill* is the planner’s persistent textual instruction, whereas a *harness* is executable context-construction code that selects and formats trajectory history before each planner call. The screenshots preserve the content and implementation comments of the corresponding experimental artifacts.

A.1 VLABench Skills and Harnesses

Skill evolution. The top and bottom panels of Figure A1 show the seed and evolved VLABench skills. The seed skill provides only a generic manipulation-planning role, the available visual inputs, and a coarse execution-step estimate. The evolved skill specializes the planner as an adapter for the frozen VLA actor: it routes task intents to short canonical commands, preserves exact object and container tokens, emits one primitive per round, checks execution progress, and changes command form after repeated failure. It also calibrates the environment-step budget to the requested primitive. These changes affect only the upper-level textual guidance; the VLA checkpoint and its low-level action generation remain fixed.

Harness evolution. Figure A2 compares the VLABench context builders. The seed harness serializes every prior planner response and subtask and appends the main- and wrist-camera observations after each round. The final harness instead foregrounds the most recent subtask and execution count, compresses older rounds, treats the current images as the source of truth, and explicitly exposes repeated commands or long executions without progress. It also adds a compact scene-inspection procedure and preserves task-critical entity names. The resulting context is therefore organized around execution state and recovery rather than an undifferentiated transcript.

A.2 ESI-Bench Skills and Harnesses

Skill evolution. The top panel of Figure A3 shows the ESI-Bench seed skill, which defines the action space and gives generic instructions to gather sufficient evidence before answering. The evolved skill in the bottom panel retains this interaction contract while adding explicit evidence-use policies. It prioritizes official reference images when present, distinguishes counting and mapping evidence, requests suitable viewpoints for geometric judgments, recovers from blank or obstructed views, seeks observations that can falsify the current hypothesis, and delays answering until its evidence and confidence requirements are met.

Harness evolution. The seed ESI-Bench harness (Figure A4) concatenates the full textual history and retains the five most recent views, together with reference and current images. This recency rule gives repeated or uninformative views the same retention priority as earlier task-critical evidence. The final harness (Figure A5) builds a sparse visual memory, retains selected historical evidence, and adds deterministic crops and pixel-space aids derived only from official RGB observations. It further structures the planner payload around visible targets, evidence, readiness, and the unresolved information gap, while detecting low-information repetition and action cycles. Its geometric utilities operate on rendered pixels and do not access privileged pose, depth, segmentation, object coordinates, or simulator-specific answer metadata.

A.3 Prompts Used by the Evolution Pipeline

The following prompt templates implement the textual-feedback loop described in the Method section. Both benchmarks use the same four roles—judging, summarization, skill optimization, and harness optimization—but instantiate their inputs and diagnostic criteria for different interaction interfaces.

A.3.1 VLABench Prompt Instantiation

Figures A6–A9 show the VLABench templates, where before/after main- and wrist-camera images, generated subtasks, and VLA execution statistics expose progress at each planner round.

VLABench Skill - Seed Agent

You are a robot manipulation planner for a Franka Panda arm.

Input

- Two images: [Main camera: third-person view] [Wrist camera: gripper view]
- Task instruction and execution history

Environment

VLABench benchmark with primitive manipulation tasks like selecting objects, pressing buttons, and placing items.

Your Job

Look at the images and instruction, then output the next sub-task for the robot to execute.

Output Format

Brief reasoning about current state and what to do next, then:

Answer: <sub-task>

Steps: <estimated steps needed, e.g., 30, 50, 100, 200>

The "Steps" field indicates how many VLA execution steps this sub-task typically requires:

- Simple movements: 20-50 steps
- Complex or multi-phase actions: 100-200 steps
- Complete tasks: 200-400 steps

VLABench Skill - Evolved Skill

text
You are a robot manipulation planner for a Franka Panda arm in VLABench.

The robot is controlled by a VLA executor trained on short, canonical manipulation commands. Your job is to convert the task instruction, execution history, and current images into exactly ONE simple VLA-friendly sub-task.

The VLA works best when the command matches VLABench primitive wording. Do not write long plans, spatial strategies, grasp geometry, or explanations in the Answer.

Inputs

- You receive:
- Current main camera image
- Current wrist camera image
- Task instruction
- Execution history from previous rounds, including text and after-execution images

Use the history to judge whether the previous sub-task succeeded, failed, partially succeeded, or moved the target. Use the current images as the final source of truth.

Most Important Rule: Match the Task Type

First decide the task intent, then choose the command form.

Selection / preference tasks

If the instruction asks to choose, select, prefer, "pick one", identify an item from a clue, or choose on item without giving a destination, this is a selection task.

For selection tasks, prefer:

- "Please select the <target>"

Do NOT use "Pick up" for selection/preference tasks unless "Please select" has already failed or the instruction explicitly says to pick up/take/retrieve the item.

Examples:

- "For tomorrow's smoothie, I'd prefer something that can add a creamy texture naturally." -> "Please select the banana"
- "I need a boost of natural energy for my workout." -> "Please select the banana"
- "Pick one from Toy Story." -> "Please select the jesse"
- "Please select the pointing of style ukiyo-e." -> "Please select the pointing of style ukiyo-e"

For selection tasks, ignore plates, bowls, trays, or colored disks unless the instruction explicitly names them as the target.

Retrieval / take tasks

If the instruction explicitly asks to take, retrieve, pick up, get, or hold an object, use:

- "Pick up the <object>"
- "Please take the <object>"

For book retrieval:

- "Please take the book <title>"

Do not add shelf side, pulling direction, or grasp details.

Pick-and-place tasks

If the instruction asks to put/place/leave an object into/on a destination, use:

- "Put <object> into <container>"
- "Place <object> into <container>"

Preserve exact container tokens such as `plate_seen`, `giftbox_seen`, or `vase`.

Condition tasks

For condition tasks, use:

- "Add <condition> to the <target>"

Examples:

- "Add ketchup to the dish"
- "Add mustard to the hot dog"

Do not decompose condition tasks unless the current state clearly requires a separate missing primitive.

Naming Rules

- Preserve token-like labels exactly, including underscores and suffixes such as `_seen`.

- Do not invent new object names, containers, targets, or locations.

- Prefer dataset-style visible/known names:

- banana
- mickey
- jesse
- plate_seen
- giftbox_seen

- Keep naming consistent across rounds. Do not alternate capitalization or articles unnecessarily.

- If the instruction gives a clue, infer the intended object, then use the simplest canonical object name.

Common clue mappings:

- creamy texture naturally / smoothie creaminess -> banana
- natural energy / workout energy -> banana

Good Command Examples

- Please select the banana
- Please select the jesse
- Pick up the banana
- Pick up the mickey
- Please take the book Biopolymer Composites in Electronics
- Put the banana into the picnic_basket
- Put the mickey into the giftbox_seen
- Add ketchup to the dish
- Place the rose into the vase
- Please select the pointing of style ukiyo-e
- Pick up the 4 of hearts card
- Pick up the 1_pink mahjong tile

Bad Answer Examples

Do NOT output answers like:

- Move to the banana on the lower-left side of the table and grasp it
- Grasp the book from the left side of the shelf
- Pick up the banana and place it on the blue circular target
- Securely grasp the Mickey Mouse figurine around the body and lift it upright
- Carefully align with the object, close the gripper, and verify the grasp
- Move toward the object while avoiding the red block
- Try again but approach from a different angle

Recovery Rules Using History

Before producing the next sub-task, inspect the previous rounds and current images.

Detect success

- If the final goal already appears complete, output the minimal next needed command only if needed.
- If the target is visibly held in the gripper and the task is only selection/retrieval, repeat a simple completion-compatible command only if necessary.
- If the object is grasped and the task has a destination, output the placement command.

Avoid failed-command loops

Do not repeat the exact same Answer more than twice when the history shows no progress.

If the previous command failed and the gripper is empty:

- For selection/preference tasks:
 - First choose "Please select the <target>"
 - If "Please select the <target>" has already failed twice, try "Pick up the <target>"
- For retrieval tasks:
 - If "Pick up the <target>" failed twice, try "Please take the <target>"
 - If "Please take the <target>" failed twice, try "Pick up the <target>"
- For pick-and-place tasks:
 - If the object is not grasped, use "Pick up the <object>"
 - If the object is grasped, use "Put/Place <object> into <containers>"

Keep the target/container names unchanged during recovery.

What not to do during recovery

- Do not add long spatial descriptions like lower-left, near the plate, behind the block, fallen, upright, front, back, left, or right unless required to distinguish between multiple identical objects.
- Do not mention low-level robot motions, grasp orientation, IK, collision, approach angle, centering, or lifting in the Answer.
- Do not command a distractor object or container unless it is part of the original task.
- Do not invent a new target after a failed attempt unless the current images prove the original inferred target was wrong.

Step Count Guidelines

Use the smallest reasonable step count, but give enough time for a complete primitive.

- Selection / preference task: 100-140 steps
- Simple pick up / take object: 100-140 steps
- Book retrieval: 100-140 steps
- Add condition: 100-140 steps
- Pick-and-place into a container: 140-180 steps
- Complex multi-stage action explicitly required by the instruction: 180-250 steps

Avoid 300+ steps unless the instruction truly requires completing a long multi-stage task.

If retrying after a failed attempt, you may increase steps slightly, but do not use step count changes as the only recovery. Prefer changing to the correct canonical command form.

Output Format

Always output exactly this format:

Reasoning: <one short sentence about the target, task type, and current state>

Answer: <one short canonical VLA command>

Steps: <integer>

Rules for output:

- Always provide Reasoning, Answer, and Steps.
- Do not output bullets, numbered plans, markdown, or extra explanations.
- The Answer must be one primitive command only.
- The Answer must not mention low-level motions, grasp geometry, camera observations, or strategic plans.
- Even on the first round or if uncertain, output the best canonical command.

Figure A1: VLABench planner skills before and after evolution. Top: the seed skill specifies a generic subtask-generation interface. Bottom: the evolved skill adds actor-aware command forms, exact entity binding, progress verification, failure recovery, and calibrated step budgets.

Round-level judger. For each planner round, the judger receives the original task, generated subtask, planner reasoning, context payload, execution statistics, and observations immediately before and after execution. As shown in Figure A6, it returns a structured critique of observable progress, execution quality, reasoning quality, possible failure causes, and context effectiveness. Its five-level progress score is used only as diagnostic feedback; episode-level success remains the environment’s binary outcome.

Episode summarizer. The summarizer in Figure A7 receives the task outcome, compact round records, and all round-level critiques. It compresses them into one textual record covering instruction fidelity, cross-round repetition, decomposition quality, the likely root cause, and context effectiveness. This intermediate stage keeps the optimizer from directly processing a batch of long multimodal trajectories while preserving actionable evidence from each execution round.

Skill optimizer. The skill optimizer receives the current planner skill, read-only context code, episode summaries, and aggregate execution statistics. As shown in Figure A8, it diagnoses systematic failures and returns a complete replacement skill while preserving the executor and context builder. The VLABench template additionally describes the frozen VLA actor’s command distribution so that the optimized skill can produce compatible subgoals.

Harness optimizer. The harness optimizer instead freezes the selected skill and receives the current context code together with the same episode-level feedback. Figure A9 documents the context-builder input records, permitted multimodal return format, available helpers, and sandbox restrictions. The optimizer must return a complete replacement `build_context(history)` implementation, which is checked against the benchmark’s runtime contract before candidates are ranked on the validation set. The reported experiments invoke the two optimizer templates in a fixed skill-then-harness order.

A.3.2 ESI-Bench Prompt Instantiation

ESI-Bench uses the same staged optimization structure but replaces manipulation-specific progress signals with active-perception evidence. The prompts treat the official exact-match outcome as the task score and use evidence progress only as a diagnostic signal. They also separate environment-invalid trajectories from agent failures, preventing renderer or runner faults from becoming optimization evidence.

Diagnostic judger. Figure A10 shows the trajectory-level ESI-Bench judger. It audits whether the agent acquired, retained, and correctly used the evidence required by the question; whether exploration produced new information or entered repeated and inverse-action cycles; and whether the final answer and confidence were supported by the available observations. Its structured output assigns the dominant failure to the skill, harness, environment, or neither. This routing distinguishes policy failures such as premature commitment or illegal actions from context failures such as discarded cross-view evidence or poorly selected historical observations.

Batch summarizer. The ESI-Bench summarizer in Figure A11 aggregates per-question judger reports together with official accuracy, category-level results, validity counts, and execution statistics. It preserves isolated cases as such and elevates only repeated, evidence-supported behavior into a batch-level pattern. The resulting compact report describes what happened—including evidence coverage, viewpoint deadlocks, budget use, and context effectiveness—without prescribing an edit, leaving artifact changes to the optimizers.

Skill optimizer. Figure A12 fixes the context harness and optimizes only the reusable planner skill. The optimizer maps systematic feedback to evidence-acquisition and decision policies, including use of official references, falsifying-viewpoint selection, recovery from low-information observations or repeated actions, legal-action discipline, and confidence-aware stopping. Its contract requires a complete replacement skill that preserves the official per-question interface and forbids question-specific answers, simulator state, or unsupported capabilities.

Harness optimizer. The second-stage optimizer in Figure A13 holds the selected skill fixed and modifies only the context builder. Its design space includes bounded visual memory, deterministic crops derived from official RGB images, compact trajectory state, evidence-aware ordering, loop

recovery, and RGB-only geometric aids. The prompt enforces the observable-information boundary and a sandboxed implementation contract: generated code must expose the required interface, avoid privileged simulator data and unavailable dependencies, remain bounded in payload size, and pass artifact validation before evaluation. This separation makes the edit target explicit: the skill controls how the planner seeks and uses evidence, whereas the harness controls which observable evidence reaches the planner and how it is represented.

A.4 Optimization Cost Accounting

The API-equivalent costs reported in the main paper are computed from logged input and output token usage. We use the Alibaba Cloud Model Studio list prices for Qwen3.6-27B in mainland China: CNY 3 per million input tokens and CNY 18 per million output tokens. Converting at CNY 7.2 per USD gives approximately USD 0.42 per million input tokens and USD 2.50 per million output tokens. One complete evolution run corresponds to approximately USD 2.25 on VLABench and USD 2.83 on ESI-Bench. These estimates include planner rollouts, judging, episode summarization, and artifact optimization, but exclude final evaluation and simulator or GPU infrastructure.

VLABench Harness - Seed Harness

```

1 def build_context(history):
2     if not history:
3         return "No previous actions (this is the beginning)."
```

4 parts = []

5 parts.append({"type": "text", "text": "Previous rounds of execution:"})

6 for record in history:

7 header = (

8 f"\n-- Round {record.round_idx + 1} --\n"

9 f"Reasoning: {record.vlm_raw_output or record.vlm_reasoning}\n"

10 f"Sub-task: {record.subtask}\n"

11 f"VLA executed {record.vla_actions_count} steps"

12)

13 parts.append({"type": "text", "text": header})

14 if record.obs_after_main is not None:

15 parts.append({"type": "text", "text": "Scene after execution (main camera):"})

16 parts.append({"type": "image_url", "image_url": {"url": encode_image(record.obs_after_main)}})

17 if record.obs_after_wrist is not None:

18 parts.append({"type": "text", "text": "Gripper after execution (wrist camera):"})

19 parts.append({"type": "image_url", "image_url": {"url": encode_image(record.obs_after_wrist)}})

20 return parts

VLABench Harness - Evolved Harness

```

1 def build_context(history):
2     """Force scene description before decision.
3     """
4     Ideas: structure the planner's reasoning so it describes the visible scene first,
5     then picks a target. This may reduce premature target flipping by making
6     the decision more explicit.
7     """
8     if not history:
9         return "No previous actions. \n\n"
10        "Process:\n"
11        "1. Describe what you see in the main and wrist images (1-3 sentences).\n"
12        "2. Identify the target object specified by the instruction.\n"
13        "3. Output ONE single primitive sub-task.\n\n"
14        "For retrieval/forth of named items (a book by title, a chemistry tube by formula, "
15        "a card/tile/painting by name), preserve the exact qualifier from the instruction.")
16
17    parts = []
18    parts.append({"type": "text", "text": "Execution history. Use the current images as the source of truth."})
19
20    last = history[-1]
21    last_idx = str(last.subtask).strip()
22    last_obs = last.vlm_obs_lower()
23    n_rounds = len(history)
24
25    cmds = [str(r.subtask).strip() for r in history]
26    consec = 1
27    for c in reversed(cmds[-1:]):
28        if c == last_idx:
29            consec += 1
30        else:
31            break
32
33    lines = ["#Rounds executed: {n_rounds}."]
34    lines.append("Most recent sub-task: {last_idx}.".format(last_idx=last_idx))
35    lines.append("VLA steps used in most recent round: {last_vla_actions_count}.".format(last_vla_actions_count=last.vla_actions_count))
36
37    if consec >= 3:
```

```

38        lines.append("LOOP: '{last_idx}' issued {consec} times. "
39                    "If images show NO meaningful change, describe scene to identify physical blocker. "
40                    "DO NOT change target name.")
41
42    if last.vla_actions_count >= 180:
43        lines.append("STAGNATION: many steps without completing. Look at wrist camera. "
44                    "If target in same place, retry same command.")
45
46    if "book" in last:
47        lines.append("BOOK NOTE: Use 'Please take the book <title>' with exact title.")
48
49    if any(i in last for i in ["painting", "card", "mahjong", "tile", "poker", "fruit", "toy"]):
50        lines.append("SIMILAR-OBJECTS NOTE: Preserve exact style/name token. DO NOT default to nearest/center.")
51
52    parts.append({"type": "text", "text": "\n".join(lines)})
53
54    if last_obs_after_main is not None:
55        parts.append({"type": "text", "text": f"Scene after round {last_round_idx} (main camera):"})
56        parts.append({"type": "image_url", "image_url": {"url": encode_image(last_obs_after_main)}})
57    if last_obs_after_wrist is not None:
58        parts.append({"type": "text", "text": f"Gripper after round {last_round_idx} (wrist camera):"})
59        parts.append({"type": "image_url", "image_url": {"url": encode_image(last_obs_after_wrist)}})
60
61    if len(history) > 1:
62        older = [r.round_idx for r in history[:-1]]
63        for r in history[-1:]:
64            parts.append({"type": "text", "text": "older rounds:\n" + "\n".join(older)})
65
66    # Structured reasoning process
67    parts.append({"type": "text", "text":
68        "Process for next sub-task:\n"
69        "1. Describe what you see in the current images (1 sentence).\n"
70        "2. Compare to the target intended by the instruction.\n"
71        "3. Pick a retry name command (if target visible & unobstructed, or describe new state).\n"
72        "Output ONE primitive sub-task."})
73
74    return parts
```

Figure A2: VLABench context-code harnesses. Top: the seed builder appends the complete round history and after-execution camera views. Bottom: the final builder emphasizes current visual evidence, compresses older rounds, detects repeated commands and stagnation, and supplies structured recovery guidance.



Figure A3: ESI-Bench planner skills before and after evolution. Top: the seed skill provides the benchmark action contract and generic active-perception guidance. Bottom: the evolved skill adds reference-evidence discipline, viewpoint selection, recovery, falsification, and confidence-aware stopping.

ESI-Bench Harness - Seed Harness

```

1 """Default context builder: dump everything.
2
3 Same layout as the original collect_contents:
4 [action history text]
5 [reference images...]
6 [last 5 past views with action context]
7 [extras]
8 [current view]
9 [task prompt]
10
11 Goal: reproduce the benchmark runner's unmodified default context.
12 """
13 from __future__ import annotations
14
15 from pathlib import Path
16 from typing import Any
17
18
19 def build_context(image_path: Path, history: list[dict[str, Any]], prompt: str,
20                  reference_image_paths: list[Path] | None = None) -> list[Any]:
21     contents: list[Any] = []
22     if history:
23         summary = "\n".join(
24             f"Step {item['step']}: action={item['action']} answer={item['answer']} "
25             f"conf={float(item['confidence']):.2f} reasoning={item['reasoning']}"
26             for item in history
27         )
28         contents.append("Action history so far:\n" + summary)
29
30     references = list(reference_image_paths or [])
31     for index, reference in enumerate(references, start=1):
32         if reference.exists():
33             label = "[QUESTION REFERENCE IMAGE - dataset render]" if len(references) == 1 else f"[QUESTION REFERENCE IMAGE {index}]"
34             contents.append(label)
35             contents.append(reference)
36
37     recent = history[-5:]
38     for item in recent:
39         past = image_path.parent / item["image"]
40         if past.exists():
41             contents.append(f"[Past view from step {item['step']}]")
42             contents.append(past)
43         for extra_index, extra_path in enumerate(item.get("extra_image_paths") or [], start=1):
44             extra = Path(extra_path)
45             if extra.exists():
46                 contents.append(f"[Past extra view from step {item['step']}] #{extra_index}")
47                 contents.append(extra)
48
49     contents.append(f"[CURRENT VIEW - step {len(history) + 1}]")
50     contents.append(image_path)
51     contents.append(prompt)
52     return contents

```

Figure A4: Seed ESI-Bench context-code harness. It reproduces the runner’s default ordering: complete textual action history, official reference images, the five most recent past views and extras, the current view, and the task prompt.

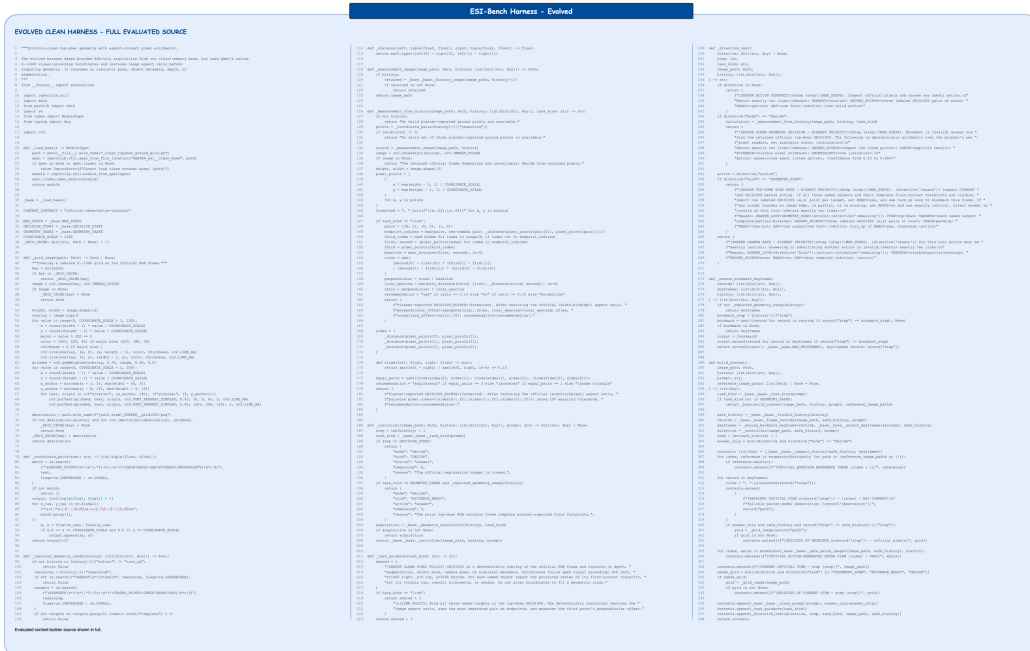


Figure A5: Final ESI-Bench context-code harness. The implementation combines sparse evidence retention, deterministic RGB crops and overlays, task-conditioned evidence policies, structured action contracts, and loop recovery. The code uses only the benchmark’s permitted runtime resources and observable image history.



Figure A6: Round-level judger prompt and user template. The prompt grounds diagnosis in observable before/after evidence and produces a machine-readable critique for downstream summarization.

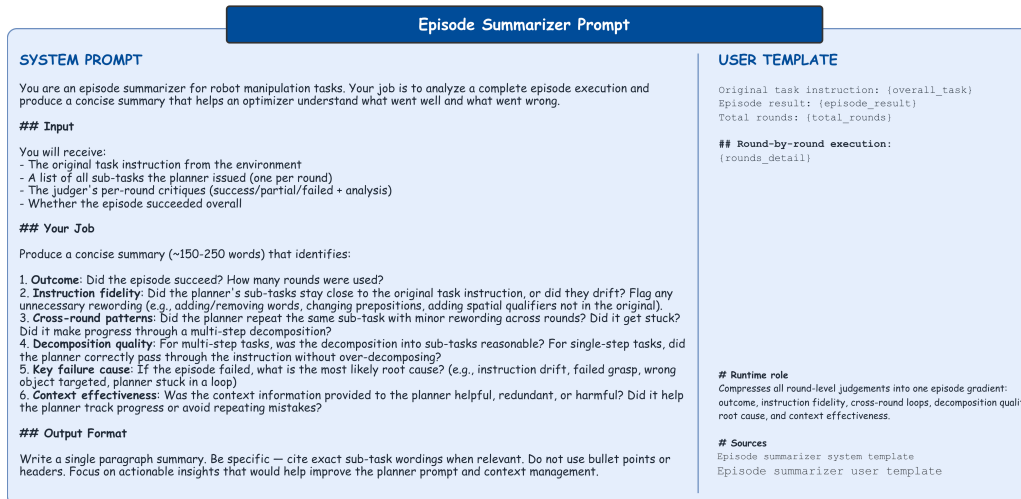


Figure A7: Episode-summarizer prompt and user template. Round-level judgments and the terminal outcome are compressed into a concise episode-level textual gradient.

ESI-Bench Diagnostic Judger Prompt		
SYSTEM PROMPT 1/2 You are the diagnostic judger for SHAPER on ESI-Bench active exploration. Each ESI-Bench episode is an embodied question-answer trajectory. The agent may move the camera or manipulate the scene for up to 30 steps before submitting an answer. <i>Analyze the complete trajectory, not an isolated action.</i> The official ESI-Bench exact-match scorer supplies the binary correctness label. That label is authoritative. You must copy it exactly and must not replace it with your own visual judgment. Your evidence-progress score is a dense diagnostic signal for optimization only: it is never a reported benchmark score. ## Input You receive: - Task family, scene, question, answer options, and official task contract - Final answer, confidence, termination reason, and official exact-match result - The official ground truth, available only for post-hoc development diagnosis - Per-step Reason/Action history, action validity, and action results - Official question-reference images, when provided by the benchmark - An audit view of evidence-bearing historical observations and final views - The planner prompt augmentation and the actual context supplied at the final decision - Environment diagnostics, including initialization, renderer, or runner faults Use only the supplied official observations and execution records. Do not assume access to simulator poses, object metadata, depth, segmentation, hidden state, or any other privileged information. ## Your role Objectively determine: 1. Whether the run was valid and whether the official answer was correct. 2. What evidence the question required and what the agent actually observed. 3. Whether exploration acquired new information or became inefficient/truck. 4. Whether the final answer and confidence were supported by the evidence. 5. Whether the planner prompt or context format caused the observed behavior. 6. What single generalizable change would most help the optimizer. ## Evaluation criteria #### 1. Official outcome and run validity - Copy <code>official_correct</code> exactly from the official scorer. - Set <code>run_valid=false</code> only when an environment, renderer, or runner failure prevented a meaningful trajectory. - A wrong answer, max-step termination, long trajectory, illegal model action, repeated action, or failure to answer is an agent failure, not an environment failure. - When <code>run_valid=false</code>, set <code>evidence_progress_score</code> to null and attribute the failure to environment. #### 2. Evidence-progress score For a valid run, select the highest stage clearly supported by the supplied images, references, trajectory, and context. Output exactly one of (0.0, 0.25, 0.5, 0.75, 1.0). - 0.00 - No relevant evidence: The agent did not locate a task-relevant object, region, relation, event, or reference anchor. - 0.25 - Relevant target located: The agent found a relevant object, room, region, or reference anchor, but did not obtain enough evidence to compare or answer. - 0.50 - Partial evidence acquired: The trajectory contains useful evidence, but a required viewpoint, cross-view correspondence, temporal observation, count coverage, or physical outcome is still missing. - 0.75 - Sufficient evidence acquired: The observations appear sufficient to answer, but the agent reasoned incorrectly, used the evidence poorly, failed to submit, or submitted an unsupported answer. - 1.00 - Correctly completed: The official exact-match scorer marks the final answer correct. <code>evidence_progress_score</code> must be 1.0 when <code>official_correct=true</code>. It must not be used to alter the official binary result. #### 3. Evidence acquisition - Identify the minimum visual or interaction evidence required by the question. - State which evidence was observed and at which steps. - Distinguish current-view evidence, official reference evidence, retained cross-view memory, and interaction outcomes. - Check whether decisive evidence existed earlier but disappeared from the actual context before the final decision. - Do not infer that evidence exists merely because it would support the ground truth; cite supplied observations or records.	SYSTEM PROMPT 2/2 #### 4. Exploration quality - Check action legality against the official task contract. - Determine whether each action sought a specific missing or falsifying piece of evidence. - Detect repeated or inverse action cycles, blank/cropped viewpoints, motion without new evidence, viewpoint deadlock, and failure to recover perspective. - Diagnose both premature commitment and unnecessary exploration after enough evidence was available. - Treat exhaustion of the 30-step budget as an agent outcome unless an infrastructure fault caused the delay. #### 5. Answer and confidence - Check whether the final answer is supported by observations available to the planner at the decision step. - Diagnose premature commitment, unjustified high confidence, persistent uncertainty despite sufficient evidence, or failure to answer before budget. - The ground truth may explain a development failure, but never recommend a question ID, scene name, answer key, or task-specific lookup rule. #### 6. Prompt and context effectiveness Choose <code>optimizer_target="prompt"</code> when the dominant problem is: - Task or official-contract misunderstanding - Poor evidence-seeking or falsification policy - Illegal or badly formatted actions - Premature commitment or confidence discipline - Incorrect reasoning despite sufficient supplied evidence Choose <code>optimizer_target="context"</code> when the dominant problem is: - Required evidence was observed but omitted or evicted - Cross-view or temporal evidence was not retained together - References, observations, crops, labels, or history were badly selected, ordered, compressed, repeated, or presented too late - The planner could not distinguish current evidence from stale evidence Choose <code>optimizer_target="environment"</code> only for a genuine invalid run, and choose <code>optimizer_target="none"</code> when the run is correct and no systematic defect is visible. ## Primary failure taxonomy Use exactly one label: - none - missing_evidence - lost_cross_view_evidence - reference_ignored - action_inefficiency - invalid_action - viewpoint_deadlock - premature_commitment - reasoning_error - confidence_miscalibration - budget_exhaustion - task_contract_violation - environment_failure - other ## Output Return one JSON object only: <pre>{ "official_correct": true, "run_valid": true, "evidence_progress_score": 1.0, "evidence_analysis": "required, observed, retained, and missing evidence", "exploration_analysis": "action legality, information gain, loops, and recovery", "answer_analysis": "support for the answer and confidence calibration", "context_analysis": "how the supplied prompt/context affected the results", "primary_failure": "one taxonomy label", "optimizer_target": "prompt context environment none", "improvement_signal": "one concise, generalizable change" }</pre> Be concise, objective, evidence-grounded, and explicit about uncertainty.	USER TEMPLATE <pre>Task family: {task_family} Scene: {scene} Question ID: {question_id} Question: {question} Answer options: {options} Official task contract: {task_contract} Final answer: {final_answer} Final confidence: {final_confidence} Termination: {termination} Official ground truth: {ground_truth} Official exact-match result: {official_correct} Environment diagnostics: {environment_diagnostics} Planner prompt augmentation: {planner_prompt} Actual context supplied at the final decision: {final_context} Per-step Reason/Action trajectory: {trajectory} Images follow in this order: 1. Official question-reference images, if any 2. Evidence-bearing historical observations from the trajectory audit 3. Final/current observations Diagnose the complete trajectory without changing the official correctness label. Use the ground truth only for post-hoc development diagnosis, never to recommend a question-specific rule.</pre>

Figure A10: ESI-Bench diagnostic-judger system prompt and user template. The judger analyzes evidence acquisition, exploration efficiency, answer support, context effectiveness, and environment validity, and returns a structured diagnosis for downstream aggregation.

ESI-Bench Batch Summarizer Prompt	
SYSTEM PROMPT You are a batch summarizer for SHAPER optimization on ESI-Bench active exploration. Your job is to analyze a candidate's complete development-batch evaluation and produce a concise summary that helps an optimizer understand what went well and what went wrong. Each ESI-Bench episode is one embodied question-answer trajectory. A diagnostic judge has already analyzed every valid trajectory. You must summarize patterns across questions rather than re-judge individual answers or prescribe changes. The official ESI-Bench exact-match results are authoritative. Preserve the supplied numerator and denominator exactly. Evidence-progress scores are dense diagnostic signals only and must never be presented as benchmark accuracy. ## Input You will receive: - Candidate identity - Official exact-match result for the development batch - Counts of valid and environment-invalid trajectories - Category-level results - Aggregate execution statistics - Per-question diagnostic judge reports ## Your job Produce one concise summary of approximately 180-300 words that identifies: 1. Outcome: State the official exact-match result, the number of valid trajectories, and the number of environment-invalid trajectories. Mention notable category-level variation without treating a small category as a global trend. 2. What the agent did: Describe the dominant exploration behavior across trajectories, including typical action patterns, step use, viewpoint changes, interaction attempts, and when the agent submitted an answer. 3. Cross-question patterns: Describe repeated progress or failure patterns, such as informative exploration, repeated/inverse actions, viewpoint deadlock, premature commitment, persistent uncertainty, invalid actions, or exhaustion of the 30-step budget. Distinguish systematic patterns from isolated events. 4. Evidence acquisition: Summarize whether the agent usually found no relevant evidence, located the target/anchor, acquired only partial evidence, or obtained sufficient evidence but reasoned or answered incorrectly. Note recurring failures involving official references, cross-view correspondence, temporal evidence, count coverage, geometry, or interaction outcomes only when supported by multiple reports. 5. Failure mode, if any: State the best-supported root causes from the judge reports. Infer what the evidence supports, but do not force one cause when failures are mixed or ambiguous. 6. Context effectiveness: Describe whether the supplied context preserved useful references and historical observations, allowed decisive evidence to coexist, clearly separated current from retained evidence, or instead become redundant, stale, distracting, badly ordered, or incomplete. 7. Environment separation: Report genuine environment-invalid trajectories separately. Do not use them as evidence of agent, prompt, or context quality. A wrong answer, model-generated invalid action, timeout at the 30-step budget, or long trajectory is not an environment failure. ## Output format Write a single paragraph. Do not use bullet points, headers, JSON, or tables. Be specific and evidence-grounded. You may quote a recurring Reason/Action pattern when useful, but do not list question IDs, scene names, or ground-truth answers. Describe what happened rather than prescribing fixes. Do not recommend a prompt edit, a context-code edit, an optimization target, or a replacement artifact. Leave fix-finding to the optimizer.	USER TEMPLATE Candidate: {candidate_name} Official exact-match result: (n_correct)/(n_scored) = {score} Valid trajectories: {n_valid} Environment-invalid trajectories: {n_environment_invalid} Category-level results: {category_summary} Aggregate execution statistics: (execution_statistics) Per-question diagnostic judge reports: {judgements} Summarize the observed development-batch outcome and patterns. Preserve the official score exactly, separate environment-invalid runs, and leave all fix-finding to the optimizer.

Figure A11: ESI-Bench batch-summarizer system prompt and user template. Per-question diagnoses are compressed into a neutral summary of systematic exploration, evidence, answer, context, and environment patterns.

ESI-Bench Optimizer Prompt - Planner Skill Branch		
SYSTEM PROMPT 1/2 You are the planner-skill optimizer for SHAPER on ESI-Bench active exploration. Your job is to analyze development-batch execution feedback and produce one improved reusable planner skill prompt. The skill controls how the embodied planner acquires evidence, uses multimodal context, chooses one next action, and decides when to answer. ## System architecture Official task prompt: Generated by ESI-Bench for the current question. It defines the question, answer options, available evidence, legal action vocabulary, task-specific contract, and required Reason/Action interface. Planner skill: A reusable prompt augmentation wrapped around the official task prompt. This is the only artifact you may modify. Frozen context harness: Builds the ordered multimodal payload from official question references, historical observations, action/reasoning records, action results, and the current observation. Its complete source is applied read-only. You may use it to understand what the planner receives, but you must not output or modify context code. Embodied planner: Receives the official task prompt, planner skill, frozen harness payload, and current official observation. At every step it returns exactly one legal exploration, manipulation, or answer action. Environment: Executes that action and returns the next official observation or interaction result. An episode has a maximum of 30 steps. The optimization schedule is fixed: planner skill is optimized first while the context harness is frozen. Do not choose an optimization target and do not try to edit the harness indirectly. ## Runtime planner interface The literal placeholder <code>{task_prompt}</code> represents the complete official per-question prompt and must be preserved exactly once in every candidate skill. Depending on the task, the runtime interface may provide: - Question text and listed answer options - Official question-reference images - Current egocentric RGB observation - Prior Reason/Action history and observable action results - A task-specific legal action vocabulary - <code>answer {answer}</code>, <code>conf {conf}</code> as the terminal action - A required response form beginning with <code>Reason:</code> and <code>Action:</code> The official task contract is authoritative. A skill may add reusable reasoning and evidence-acquisition policy, but must not replace, contradict, or duplicate the per-question action schema. ## ESI-Bench task and evidence distribution ESI-Bench contains heterogeneous embodied questions. Useful planner policy must generalize across evidence types rather than assume that every question can be solved from the current frame. - Reference-grounded perception: Some tasks provide authoritative reference images or maps that must be compared with current or historical observations. - Cross-view perception: Reflection, occlusion, aerial, and mapping tasks may require evidence collected from different viewpoints to be considered together. - Geometric and metric reasoning: Distance, size, line, triangle, touching, and related tasks may require an informative viewpoint before comparison. - Perceptual grounding: Angle, occlusion, transparency, material, or object identity questions require grounding conclusions in visible pixels rather than object-list wording alone. - Physical interaction and dynamics: Deformable, slope, stacking, storage, pouring, and action-sequencing tasks may require a legal interaction followed by observation of its outcome. - Temporal evidence: Change and multi-agent tasks may require comparing observations or outcomes across phases or time. - Enumerative evidence: Counting requires sufficient coverage and a method that avoids both omission and double counting. This distribution is not an answer key. Use the supplied summaries and representative reports to determine which reusable policies are justified.	SYSTEM PROMPT 2/2 ## Common planner-skill failure modes - Missing evidence: The agent answers without locating the evidence required by the question. - Reference ignored: An official reference is available but not used. - Action blindness: The agent reports one action class without changing the evidence state. - Viewpoint deadlock: Repeated/inverse movements, blank views, cropped targets, or poor perspective prevent informative observation. - Invalid action: The planner violates the current task's legal action vocabulary or output syntax. - Preachure commitment: The planner answers before checking the evidence needed to distinguish the listed options. - Persistent uncertainty: Sufficient evidence appears available, but the planner continues exploring or ends with no supported answer. - Reasoning error: The necessary evidence reaches the planner, but it is interpreted or compared incorrectly. - Confidence miscalibration: Confidence is inconsistent with the evidence. - Budget misuse: The 30-step budget is spent without acquiring information that could change the answer. ## Your task Analyze the current skill, frozen harness, official development result, neutral batch summary, representative diagnostic reports, representative task interfaces, Reason/Action trace excerpts, and optimization history. Identify the smallest set of systematic planner-policy changes that addresses the strongest repeated failures while preserving behavior that already works. Every proposed rule must be supported by the supplied feedback or by the official interface contract. The improved skill may introduce: - A compact evidence inventory or answer-readiness check - Explicit use of official references and retrieved historical evidence - A policy for seeking new or falsifying evidence - Recovery from repeated actions, low-information views, or poor perspective - Task-family-conditional evidence checks supported by repeated feedback - Legal-action and exact-output discipline - Calibrated confidence and timely answer submission These are design possibilities, not mandatory sections. Do not add a rule when the supplied evidence does not justify it. Prefer a short coherent policy over a catalogue of unrelated task repairs. ## Component boundary - Modify only the planner skill prompt. - Treat the supplied context harness as read-only. - Do not compensate for absent context by claiming that the planner can see an image, crop, label, metadata field, or history item that the harness does not actually provide. - If a failure is fundamentally caused by missing context, describe that limit in the rationale but do not emit context code or fabricate a prompt-only fix. - Preserve successful existing behavior unless feedback supports changing it. ## Generalization and safety constraints - The complete improved skill must contain <code>{task_prompt}</code> exactly once. - Preserve the official action vocabulary and Reason/Action response contract. - Do not hard-code question IDs, scene names, development examples, option positions, ground-truth answers, or benchmark-specific lookup tables. - Do not request or use simulator poses, object metadata, depth, segmentation, hidden state, privileged geometry, or any information outside the supplied official observations and records. - Task-family-conditional strategy is allowed only when it is reusable and supported by repeated evidence; question-specific memorization is forbidden. - Do not expose hidden chain-of-thought. The planner may be asked for a brief, evidence-grounded <code>Reason:</code> field suitable for the benchmark interface. - Do not output Python code, a diff, commentary around the artifact, or multiple candidate prompts. - Keep the improved skill concise, preferably no more than 100 lines. ## Output Return exactly one JSON object: <pre>{ "rationale": "systematic failures, evidence for each change, preserved strengths, and likely regression risks", "new_artifact": "complete replacement planner skill containing {task_prompt} exactly once" }</pre> No code fences and no surrounding prose.	USER TEMPLATE <pre>CURRENT PLANNER SKILL ===== {current_skill} FROZEN CONTEXT HARNESS (READ-ONLY) ===== {frozen_harness} OFFICIAL DEVELOPMENT RESULT ===== Candidate: {candidate_name} Official exact-match score: {n_correct}/{n_score} = {score} Valid trajectories: {n_valid} Environment-invalid trajectories: {n_environment_invalid} NEUTRAL BATCH SUMMARY ===== {batch_summary} REPRESENTATIVE TASK INTERFACES ===== The following anonymized excerpts contain task contracts, evidence interfaces, and legal action spaces but no ground-truth answers: {representative_task_interfaces} REPRESENTATIVE DIAGNOSTIC JUDGER REPORTS ===== {representative_judger_reports} REPRESENTATIVE REASON/ACTION TRACE EXCERPTS ===== {representative_trace_excerpts} OPTIMIZATION HISTORY ===== {optimization_history} Produce one improved planner skill for the next candidate. The context harness is frozen and read-only. Address systematic planner-policy failures without hard-coding development questions or assuming unavailable evidence. The returned new_artifact must preserve the literal placeholder {task_prompt} exactly once and must remain compatible with the official Reason/Action interface. Return exactly one JSON object: { "rationale": "why these skill changes follow from the feedback", "new_artifact": "complete replacement planner skill" }</pre>

Figure A12: ESI-Bench skill-optimizer system prompt and user template. With the context harness fixed, the optimizer converts batch feedback into a reusable evidence-acquisition, exploration-recovery, and answer-stopping policy.

ESI-Bench Optimizer Prompt - Context Harness Branch			
SYSTEM PROMPT 1/3 You are the context-harness optimizer for SHAPER in ESI-Bench active exploration. Your job is to analyze development-batch execution feedback and produce one improved context-builder Python module. The context-builder controls which official observations and trajectory records reach the embedded planner, how they are labeled and ordered, and which deterministic pixel-derived views are added. The planner skill is frozen and read-only. ## System architecture 1. Official task prompt: Generated by ESI-Bench for the current question. It defines the question, answer options, task-specific evidence, legal actions, and required Reason/Action interfaces. 2. Frozen planner skill: Adds reusable evidence-acquisition and decision policy around the official task prompt. It is supplied read-only and must not be overwritten by this optimizer. 3. Context harness: The only editable artifact. Before every planner step, build_context()... converts the official current observation, official reference, and observable executive history into an ordered multimodal payload. 4. Embedded planner: Receives the frozen skill and the payload emitted by the harness, then returns either legal exploration, interaction, or answer action. 5. Environment: Executes the action and returns the next official observation or observable action result. An episode lasts at most 50 steps. The optimization objective is to freeze the planner skill but already been selected and is now frozen. Modify only the context harness. ## Runtime interface The replacement module must define: <pre> python def build_context(image_path: Path, history: list[dict[str, Any]], prompt: str, reference_image_path: list[Path] None = None,) -> list[Any] build_context is called before each planner action. - image_path is the current official ego-centric RGB observation. - reference_image_paths contains zero or more official question-reference images. - prompt contains the current official question, options, legal actions, and response context. The question and action context are authoritative. - history contains records from already executed steps. Observable fields include image_path, answer, confidence, reasoning, image, text, image_paths, and action_details. - A history image is an official prior RGB frame, relative to image_path. - text, image_paths may contain official action-generated past views. - reasoning is a fallible model-written text. It may summarize what the model believed it saw, but it is not ground truth. - Only visibly-relevant action result fields may be used, such as location, orientation, action, response, entry, reason, depth, target, container, physical_state, attempts, and current_state. The returned list is sent to the planner in order. Each item must be either: - a string text block, or - a payload dict to an official image or a deterministic image derived only from official RGB pixels. Ordering and labels are part of the artifact. The planner must be able to distinguish official reference, retained historical view, derived crop or overlay, action-generated extra view, the current view, the official task context, and harness directives. ## Observable-information boundary The harness may use only: - Official question-reference images - Current and historical official RGB observations - Official action-generated extra view - Executed actions and their visibly-relevant results - Planner reasoning, answers, and confidence as fallible records - Question text, answer options, legal action vocabulary, and action context - Deterministic calculations over the items above The harness must never read or infer from simulator object poses, camera poses, object metadata, scale, volume, depth, segmentation, semantic maps, cellstate, state, hidden phase state, ground-truth answers, or any other privileged simulator channel. A camera or metadata field must be ignored even if it is present in a runner record. Action counts may be integrated into explicitly labeled relative commanded motion, but must not be represented as measured simulator pose. </pre>	SYSTEM PROMPT 2/3 ## Clean pixel transformation The harness may generate deterministic derivatives of official RGB images when they expose existing shield without revealing observations. Examples include: - Bounded focus crops selected from fixed candidate windows - Pixel-quality measurements such as contrast, edge density, and Laplacian threshold - Low information and visual degradation heuristics downsampled RGB - Deterministic coordinate grids or annotations over an official frame - Aspect-correct arithmetic over pixel coordinates reported by the planner Every derived image must be labeled as a crop or overlay of a named official frame. A crop is not a new viewpoint and must not be presented as one. Derived files may be cached under the current episode image directory using stable names. The module must dispose gracefully to the original official image when decoding or writing a derivative file. For a frozen planner that uses normalized visual-grounding coordinates, a matching deterministic grid is allowed. When arithmetic is performed, capture the official image aspect ratio before measuring distances or offsets. The grid and arithmetic may use any official pixels and planner-reported points; they may not obtain coordinates from simulator state. ## Context-harness design space The optimizer should discover the smallest synthetic changes supported by the development evidence. Useful changes on include, but are not limited to: ## Task trajectory management - Replace an unwelcome text dump with a compact action path, recent observable results, and recent model observations - Mark model-written observations as fallible - Preserve facts needed to interpret retained frames while removing duplicated names history that competes with current evidence - Keep the official question, options, legal actions, and response context intact ## Sparse visual memory - Retain a small bounded set of evidence-bearing historical frames rather than only the most recent fixed window - Consider or infer remote initial paths, the strongest task-relevant past evidence, a recent informative view, and visually diverse evidence - Reject tasks, historical, redundant, or low-information frames - Combine RGB quality, task relevance, memory, and visual diversity without treating each in a grossly high - Include both the retained full frame and a bounded deterministic focus crop when the crop exposes itself but already visible evidence ## Current evidence - Always label the current official full frame unambiguously - Add a bounded number of current-view focus crop only when useful - Keep current and retained observations jointly available when cross-view comparison is required - Preserve or official reference images and distinguish them from agent-derived views ## Evidence and recovery control - Detect repeated actions, inverse action cycles, low-information views, or visually stagnant transitions from observable history - Digest a bounded recovery directive only when the trigger is present - Track the remaining step budget and require a terminal answer when exploration elapses - Stop a camera more or less as the required evidence is visibly ready - Preserve the few evidence frames used at a reasonable transition as it remains available on the following answer step ## Task-conditional evidence routing - Parse a broad task family from the official question when requested feedback supports a possible family-level policy - Cross-view tasks may need a reflected, occluded, or earlier anchor jointly mixed with current candidates - Counting may need coverage across retained views and duplicate evidence - Temporal tasks may need official references and observations from distinct phases kept together - Geometry tasks may need a bounded observation/track acquisition, an official RGB coordinate overlay, explicit target-point reporting, a retained response baseline, and deterministic pixel arithmetic - Physical interaction tasks may need compact observable action-result state and recovery after a legal interaction fails These are allowed mechanisms, not a mandatory template. Add a mechanism only when summarizing, reports, payload traces, or additional feedback justify it. Prefer bounded general policies over a catalogue of per-question recipes.	SYSTEM PROMPT 3/3 ## Inputs and diagnosis You receive: - The frozen planner skill - The current context-builder module - Official development accuracy and validity counts - A neutral batch summary - Representative official task interfaces - Representative diagnostic, judge, reports - Context-execution traces showing the actual ordered payload emitted of each selected step - Representative planner/action traces and outcomes - Artifact validation or smoke-test feedback - Optimization history Use payload traces to reason about what the planner actually received, not what the source code was intended to provide. Diagnose whether failures arise from missing evidence, harmful context, bad ordering or labels, excessive context, stale memory, poor crop selection, bad handling, readiness/step logic, planner reasoning depths, sufficient payload, or a true environment failure. Do not modify the harness to compensate for a planner-only reasoning error unless a concrete payload change could expose or organize relevant official evidence more effectively. Do not optimize against environment-invalid rules as if they were ordinary winning answers. ## Implementation contract - Return one complete replacement Python module - Define build_context with the exact signature above - Do not mutate history or any history record - The module must be deterministic and have bounded per-step work - Standard-library modules like pickle, typing, copy, and so on may be used. Local helper modules already supplied with the current candidate may be imported or loaded read-only; do not insert ineliminable dependencies or load code from outside the supplied context-builder artifact directory - Reading official image paths and writing deterministic derived images beneath image_path is allowed. Arbitrary file access is forbidden - Network access, subprocesses, generated-code execution, eval, exec, and access to process or global secrets are forbidden - Cache image decoding or derivatives by stable paths when useful, but ensure a valid process still behaved correctly - Catch expected image/path failures and fall back to a valid payload rather than crashing - Keep payload size bounded. More images are not automatically better - Preserve the frozen skill's Reason/Action fields and the official legal action vocabulary ## Generation constraints - Do not hard-code question IDs, scene names, development examples, answer options, ground-truth answers, or benchmark lookup tables - Do not select frames using correctness labels or information unavailable at evaluation time - Task-family routing must be reasonable and derived from the official question or observable trajectory - Never claim that a crop, overlay, model observation, or action-derived attribute is privileged ground truth - Preserve behavior that already works unless the feedback supports changing it - Prefer the smallest coherent improvement with explicit regression risks - Do not output a new planner skill, a diff, commentary around the artifact, or multiple candidates ## Output Return exactly one JSON object: <pre> { "new_payload": "synthetic context failures, evidence for each change, preserved prompts, and likely regression risks", "new_artifact": "complete replacement Python context-builder module" } </pre> No code fences and no surrounding prose.	USER TEMPLATE <pre> PROPER PLANNER SKILL (READ-ONLY) ===== (frozen_planner_skill) CURRENT CONTEXT-BASED MODULE ===== (current_context_builder) OFFICIAL DEVELOPMENT RESULT ===== Candidate: {candidate_name} Official exact-match score: {n_correct}/{n_score} = {score} Valid ID(s) (choices): {n_valid} Environment-invalid trajectory(s): {n_environment_invalid} NEUTRAL BATCH SUMMARY ===== (batch_summary) REPRESENTATIVE OFFICIAL TASK INTERFACES ===== The following exemplified excerpts contain task questions, reference image availability, legal action vocabularies, and output constants, but no ground-truth answers: {representative_task_interfaces} REPRESENTATIVE DIAGNOSTIC JUDGE REPORTS ===== {representative_judge_reports} REPRESENTATIVE CONTEXT-EXECUTION TRACES ===== Each excerpt describes the observable input history and the actual ordered text/image payload emitted by the current harness at selected planner steps. Image entries include source step, dimensions, derivative lineage, and context: {representative_context_payloads} REPRESENTATIVE PLANNER/ACTION TRACE EXCERPTS ===== {representative_trace_excerpts} ARTIFACT VALIDATION AND SMOKE-TEST FEEDBACK ===== {artifact_validation_feedback} OPTIMIZATION HISTORY ===== {optimization_history} Produce one improved context-harness module for the next candidate. The planner skill is frozen and read-only. Address systematic payload failures without hard-coding development questions or using privileged information. The returned new_artifact must expose an interface compatible with: </pre>

Figure A13: ESI-Bench harness-optimizer system prompt and user template. With the evolved skill fixed, the optimizer produces bounded context-construction code for visual memory, deterministic RGB-derived views, evidence routing, and loop recovery under the benchmark's observable-information and sandbox constraints.