
VISTA: Active Exploration for Self-Improvement in Embodied Agents

Zhang Zhang^{1,*}, Shengjun Zhang^{1,*,†}, Simin Huang¹, Min Chen¹, Chong Xia¹,
Xin Fei², Yueqi Duan^{1,‡}

¹Tsinghua University, ²National University of Singapore

Abstract

Agentic embodied intelligence plays a foundational role in open-ended, dynamic physical environments. Rather than treating policy learning as an end-to-end mapping from instructions to actions, recent studies reveal that embodied systems depend not only on pre-trained foundation models, but also on harness. Improving a robot harness often requires additional interaction to test hypotheses that existing execution records cannot resolve. To address this challenge, we propose VISTA, a meta-agent framework that transforms post-hoc rollout reflection into active self-exploration for embodied harness improvement. Specifically, VISTA maintains a harness DAG that records the lineage of proposed harnesses, which enables the meta-agent to retrieve prior hypotheses, build on effective mechanisms, and avoid repeatedly exploring failed directions. Each iteration begins from a retained harness version and its execution evidence. Rather than directly committing an update from these historical trajectories, the meta-agent actively explores in the environment. It autonomously generates new rollouts, probes execution failures and recovery behaviors, and iteratively modifies the harness based on the resulting evidence. The refined harness is then evaluated afresh and added to the harness DAG. We evaluate VISTA on the LIBERO-Pro, RoboCasa, and RoboTwin benchmarks. Experimental results show that VISTA improves autonomous task success through iterative harness development while keeping the underlying model parameters fixed.



Project Page: agenticei.github.io/VISTA-page/



GitHub Repo: github.com/AgenticEI/VISTA

1 Introduction

Embodied artificial intelligence has emerged as a cornerstone for enabling autonomous systems to perceive, reason, and interact seamlessly within dynamic physical environments, offering transformative potential across household assistance, healthcare logistics, and industrial automation [4, 13]. Catalyzed by multimodal foundation models and large-scale embodiment datasets, learning-based paradigms have demonstrated increasingly competitive capabilities in executing atomic manipulation tasks [3, 25, 27, 35, 37, 42, 49, 53, 58]. But scaling end-to-end policies to long-horizon, multi-stage tasks remains notoriously difficult. End-to-end models suffer severely from compounding execution errors over extended trajectories, out-of-distribution generalization under perturbations, and prohibitive sample complexity when attempting to fit multi-modal, long-tail action distributions across heterogeneous task phases [1, 23].

To overcome these fundamental limitations, the convergence of frontier large language model (LLM) agents and robotics has established embodied agentic systems as a pivotal research frontier [21, 31, 47, 48, 54]. By

* Equal contribution, † Project Leader, ‡ Corresponding author

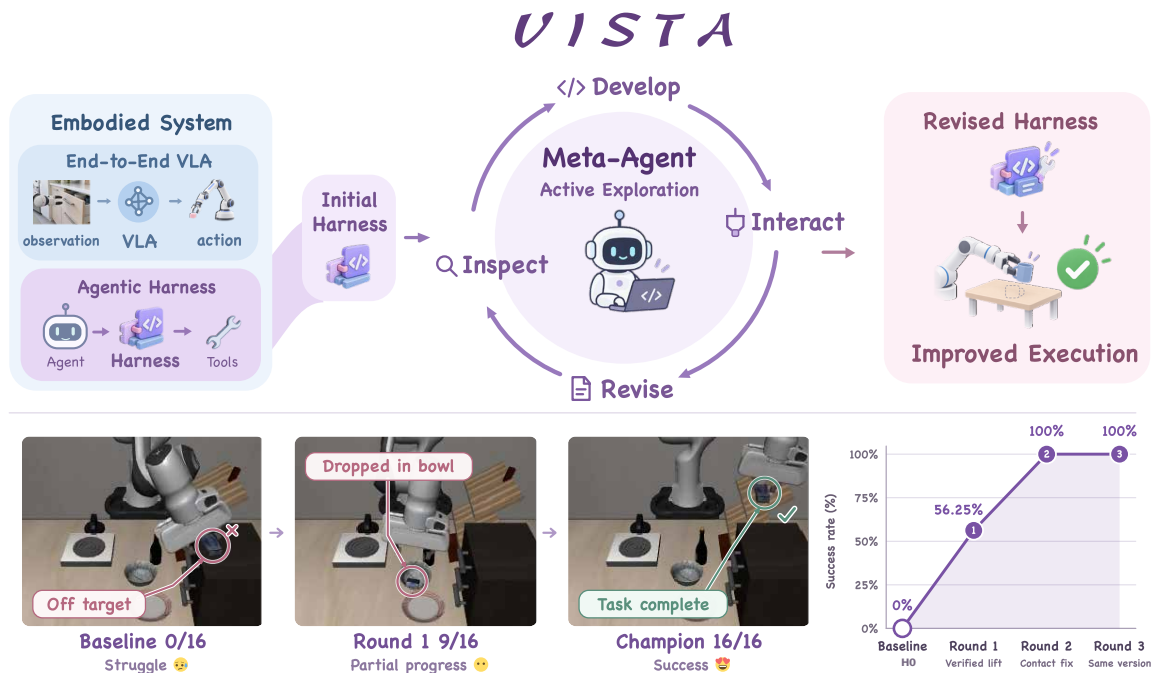


Figure 1 VISTA actively explores the environment to improve embodied-agent harnesses. Starting from an initial harness, a meta-agent inspects prior execution evidence, proposes revisions, and actively interacts with the environment to test hypotheses and refine candidate harnesses before formal evaluation. The Goal-t09 example starts from the minimal H_0 harness exposing observation and the frozen VLA policy.

coupling high-level symbolic reasoning and stateful tool orchestration with low-level control policies, agentic embodied architectures offer decisive advantages in managing long-horizon complexity: they facilitate modular hierarchical task decomposition, support active perception to resolve visual ambiguities, and enable closed-loop error diagnosis and autonomous recovery. The performance, reliability, and long-horizon coherence of such systems depend not only on their pre-trained foundation model weights, but also fundamentally on their harness [46, 52]. Existing systems construct and improve these harnesses through code generation, reusable skills, and memory-guided execution [31, 47, 48, 54]. VISTA focuses on a complementary question: how can a coding agent improve an executable robot harness by acquiring additional evidence through direct interaction with the environment? Existing records do not always contain the observations or interventions needed to test a proposed explanation of a failure.

To address this issue, we propose **VISTA**, a meta-agent framework that unifies active environmental exploration with closed-loop harness adaptation to achieve embodied self-improvement. The meta-agent controls the development process: it selects what to inspect, which public sensing and control capabilities to invoke, and which autonomous tests to request. Evidence from these interactions guides executable harness revisions. After development, the submitted harness is frozen and evaluated through fresh autonomous execution. Its code and evidence are retained in a harness DAG, allowing the meta-agent to retrieve prior hypotheses, reuse effective mechanisms, and explore from earlier candidates.

We evaluate VISTA on LIBERO-Pro [57], RoboCasa365 [36], and RoboTwin [8]. VISTA improves task success by accumulating executable mechanisms across development rounds. These results support harness development as a practical way to convert embodied experience into persistent changes in agent behavior. Our contributions are summarized as follows:

- We propose VISTA, a framework that departs from conventional handcrafted heuristics and static memory augmentation by framing the robot’s operational harness as an evolvable, executable control plane, realizing autonomous embodied self-improvement.

- We introduce an active exploration mechanism where the meta-agent empowers the embodied system to actively gather diagnostic physical evidence, linking exploratory evidence to executable harness revisions. Furthermore, we develop a Harness Graph to track the evolutionary lineage of candidate harnesses, facilitating hypothesis retrieval, component reuse, and principled branch exploration.
- We conduct comprehensive evaluations on the challenging LIBERO-Pro, RoboCasa, and RoboTwin benchmarks. Experimental results demonstrate that VISTA progressively improves manipulation success rates through accumulated experience, improving over the corresponding initial harnesses.

2 Related Work

2.1 Foundation Models for Robotic Manipulation

Low-level action policies. Foundation models for robotic manipulation can be broadly categorized according to how their semantic knowledge is connected to physical control. One line of research extends pretrained vision–language models (VLMs) into low-level policies through vision–language–action (VLA) models, which generate continuous action chunks using diffusion or flow matching [3, 12], task-space keyframes or spatial-grid actions [43], and latent action codes learned from robot trajectories or videos [5, 9]. Recent video–action and world–action models further couple action prediction with future visual dynamics [30, 40]. Robot foundation models extend these capabilities across heterogeneous tasks and embodiments [14, 39], while subsequent studies improve action initialization, tokenization, and adaptation to new tasks and embodiments [51]. Despite their strong control performance, these approaches translate general pretrained knowledge into embodiment-specific motor signals and often require additional robot trajectories when the task family or platform changes [55].

Semantic decision making and physical grounding. A complementary line of research treats foundation models as semantic decision makers rather than direct motor controllers. These models predict subgoals, keypoints, affordances, value maps, or spatial constraints, which are subsequently realized by downstream controllers [15, 21]. Although this preserves high-level semantic reasoning, it separates model intent from physical execution and depends on system-specific grounding pipelines [17, 20], as well as long-horizon planning, closed-loop feedback, and failure recovery [19]. Together, these two directions reveal a trade-off between direct physical control and semantic generalization: low-level policies offer precise execution but are closely tied to specific embodiments, whereas intermediate representations preserve semantic structure but limit physical adaptability. This trade-off motivates interfaces that combine semantic interpretability with fine-grained executable actions.

2.2 Embodied Agent Systems

Agentic control architectures. Building on these foundation-model paradigms, agentic embodied systems place a pretrained VLM within an iterative harness that translates model decisions into robot actions and feeds execution outcomes back to the model, thereby closing the perception–decision–execution loop [32, 34]. Existing systems mainly differ in the primitives through which decisions are executed. Some models generate programs over perception and control APIs [6, 18], while others select skills from predefined libraries [28, 44], steer learned policies such as VLAs with language subgoals [7, 41], or delegate decisions to symbolic planners and numerical optimizers [10, 11]. To support long-horizon interaction, these systems typically maintain task context and persistent memory [2], perform feedback-based monitoring and reflection [22], and detect failures for replanning and recovery [16].

Interfaces for embodied execution. Within agentic robot systems, interface design strongly affects control performance. While high-level, human-designed abstractions can improve reliability, composing low-level perception and control APIs remains difficult [18]. Many systems therefore expose skills, controllers, or VLAs as callable primitives [45, 54], leaving the agent to decide what to do while an opaque executor determines how to perform it [19]. Although this simplifies planning, it reduces the model’s role in fine-grained physical control. Similar compact action interfaces are used by computer-use and game agents [33, 38], simulated embodied agents [26, 29], and navigation systems [56]. Real-world manipulation still lacks

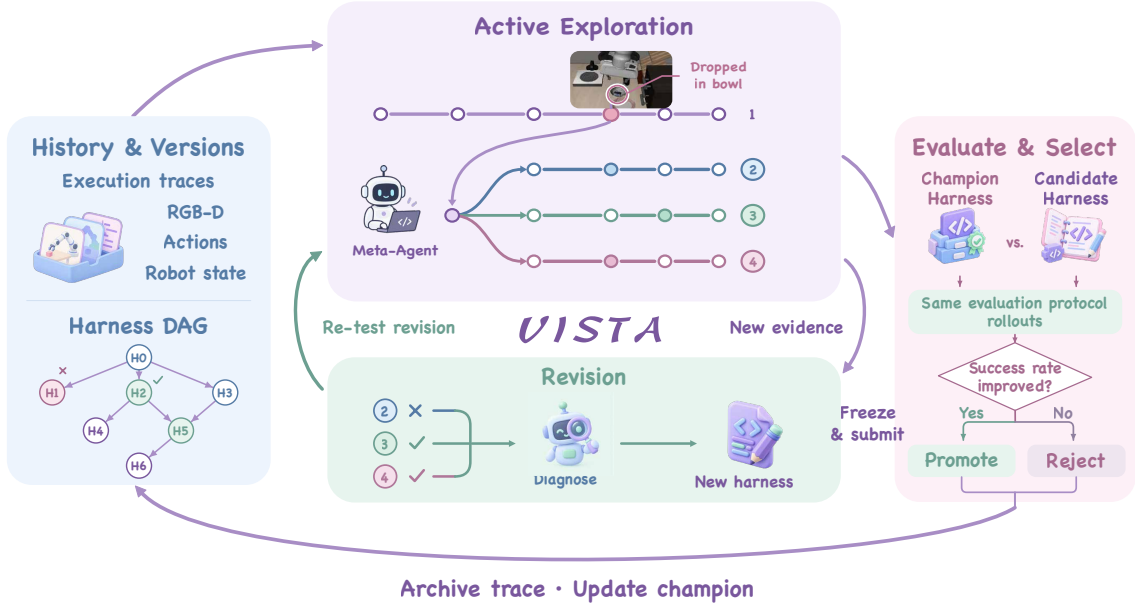


Figure 2 Overview of VISTA. The meta-agent develops an embodied harness from versioned implementations and multimodal execution evidence. Within a development iteration, it interleaves code revision, active exploration, and autonomous candidate testing. After submission, the frozen candidate undergoes a separate autonomous evaluation under the declared development conditions. The selection rule determines whether it replaces the incumbent; both promoted and rejected versions remain in the history. The inner loop acquires evidence and refines a candidate, while the outer loop selects and retains harnesses across development iterations.

a comparably simple and reusable interface because its control is high-dimensional, embodiment-specific, and spatially precise, motivating semantic action spaces with deterministic physical grounding and cross-embodiment transfer across diverse tasks and settings.

3 Method

VISTA improves an embodied agent by evolving its harness, including the instructions, executable skills, and coordination logic through which the agent perceives, reasons, and acts. The online agent has access to a language-model planner, a VLA model, and public robot capabilities. VISTA keeps the underlying model parameters fixed and optimizes how these resources are composed into autonomous behavior. Figure 2 illustrates the resulting development loop. A meta-agent, implemented as a coding agent, retrieves a harness version and its execution evidence, identifies a behavior to improve, and develops a candidate. During development, it can actively investigate the environment, probe execution failures, and revise the harness in response to newly acquired evidence. It can also request autonomous tests of intermediate versions. The submitted candidate is then frozen and evaluated in fresh autonomous episodes. A host-controlled selection rule updates the incumbent, while the candidate and its evidence remain available for subsequent iterations. Improvement is thus carried forward through both executable harness versions and the experience accumulated while developing them.

3.1 Harness DAG

Let L_θ denote the online language model, π_ϕ the VLA, and $\mathcal{C}$ the public capability interface. The interface exposes observation, perception, policy inference, and bounded robot control. A harness is a package $H = (I, \mathcal{S}, P, M, D)$, where I specifies model-facing instructions, $\mathcal{S}$ contains skill descriptions and executable implementations, P provides hooks for request construction, tool selection, and feedback processing, M

stores persistent procedural knowledge, and D declares tool bindings and capability dependencies. Skills may compose multiple perception or control operations, while policy hooks govern their use within the host’s execution loop. Harness revisions can therefore change both the operations available to the agent and how it coordinates them. The model parameters, capability interface, and evaluation rule are fixed. For a task instance x , comprising a goal and scene initialization, and execution randomness ξ , let $S(H; x, \xi) \in \{0, 1\}$ denote official autonomous task success. Our objective is

$$\max_{H \in \mathcal{H}_c} J(H), \quad J(H) = \mathbb{E}_{x \sim \mathcal{T}, \xi} [S(H; x, \xi)], \quad (1)$$

where $\mathcal{H}_c$ is the space of admissible harnesses and $\mathcal{T}$ is the task-instance distribution within the declared search scope. A frozen harness still induces closed-loop behavior: its code and persistent knowledge remain fixed during evaluation, while episode-local state and actions depend on incoming observations.

Harness development produces a directed acyclic graph (DAG) $G = (V, E)$ of immutable versions. We instantiate each harness as a Git-versioned package, with commits identifying snapshots and parent links defining lineage. Branches preserve alternative development directions, and explicit merges can combine histories. Execution records and development notes are linked to their corresponding versions. Each iteration begins from the incumbent by default, but the meta-agent may select another recorded base, including a previously rejected candidate, and inspect earlier implementations for reusable mechanisms. Retaining unsuccessful proposals makes their diagnostic findings available to later development.

3.2 Evidence-Guided Harness Revision

Each development record is tied to a frozen harness version and exact execution condition, including task initialization, repetition, randomness, and execution mode. A trajectory aligns planner decisions and tool calls with robot actions, images, and state, allowing the meta-agent to localize a failure from an aggregate outcome to a specific decision or physical execution interval.

Given a base harness and these records, the meta-agent forms a concrete hypothesis and revises the implicated components, such as instructions, observation processing, skill termination, or coordination logic. The resulting version and its supporting evidence are retained for later development. When recorded trajectories are insufficient to resolve the hypothesis, VISTA acquires additional embodied evidence through active exploration.

3.3 Active Diagnostic Exploration

Unlike static reflection, VISTA allows the meta-agent to collect new embodied evidence before submitting a candidate. It can launch a private simulator session from a frozen harness snapshot, inspect the current execution state, and invoke public sensing or control capabilities to test a specific hypothesis. These interactions provide diagnostic evidence for subsequent harness revisions rather than modifying the formal evaluation process.

After revision, the meta-agent can request fresh autonomous test rollouts to determine whether the candidate reproduces the desired behavior without intervention. Development therefore alternates among hypothesis formation, targeted interaction, harness revision, and autonomous testing until a candidate is submitted or the development budget is exhausted. Diagnostic traces and intermediate test results remain development evidence; promotion is determined only by the separate formal evaluation in 3.4.

3.4 Autonomous Validation and Selection

After submission, the host evaluates the frozen candidate through fresh episodes without meta-agent intervention. For a declared development matrix containing M task initializations and R repetitions per initialization, its empirical success is

$$\hat{J}_{\text{dev}}(H) = \frac{1}{MR} \sum_{i=1}^M \sum_{r=1}^R S(H; x_i, \xi_{i,r}). \quad (2)$$

Algorithm 1 VISTA: active exploration for harness evolution

Require: Initial harness H_{init} , development matrix $\mathcal{D}_{\text{dev}}$, budgets

- 1: Evaluate H_{init} and initialize $H^* \leftarrow H_{\text{init}}$ and development graph G
- 2: **while** budget remains and stopping rule is not met **do**
- 3: **if** H^* requires confirmation **then**
- 4: $\tilde{H} \leftarrow H^*$
- 5: **else**
- 6: Initialize development from a retained base in G (default: H^*)
- 7: **while** development budget remains and no candidate is submitted **do**
- 8: Meta-agent autonomously selects and executes the next operation from evidence
(history inspection, harness revision, live diagnosis, or autonomous testing)
- 9: Add resulting versions and evidence to G
- 10: **end while**
- 11: Validate and freeze the submitted candidate $\tilde{H}$
- 12: **end if**
- 13: Evaluate $\tilde{H}$ autonomously on $\mathcal{D}_{\text{dev}}$ from fresh initializations
- 14: Add $\tilde{H}$ and its evaluation records to G
- 15: **if** the evaluation is complete **then**
- 16: Update H^* using Eq. (3) and update its confirmation state
- 17: **end if**
- 18: **end while**
- 19: **return** H^*

Candidate and incumbent evaluations use matching task initializations, repetition settings, model configurations, and execution limits. Selection requires a complete, comparable matrix of valid autonomous outcomes. Matching incumbent evaluations may be reused. Under the success-based selection rule, the incumbent H_k^* is updated as

$$H_{k+1}^* = \begin{cases} \tilde{H}_k, & \hat{J}_{\text{dev}}(\tilde{H}_k) > \hat{J}_{\text{dev}}(H_k^*), \\ H_k^*, & \text{otherwise,} \end{cases} \quad (3)$$

where $\tilde{H}_k$ is the submitted candidate. Ties retain the incumbent. All frozen versions and their available records are retained, regardless of promotion.

Algorithm 1 summarizes the procedure. Search is bounded by the configured development and iteration budgets. When early stopping is enabled, the same selected package must achieve the required consecutive perfect formal evaluations; a confirmation iteration re-evaluates that package without another development attempt.

4 Experiments

4.1 Experiments Setup

Benchmarks. We conduct experiments on LIBERO-Pro [57], RoboCasa365 [36], and RoboTwin [8]. For LIBERO-Pro, we evaluate across *Spatial*, *Object*, *Goal*, and *Libero-10* under instruction redirection; we refer to these four suites as LIBERO-T, and additionally evaluate on the four LIBERO-S suites, which apply position perturbations that swap object and fixture initial positions. Each suite contains 10 tasks, evaluated using eight scene seeds with two repetitions per seed (16 episodes per task and 160 episodes per suite). For RoboCasa365, we consider the *Composite Seen* and *Composite Unseen* task categories. Each category contains 16 tasks, evaluated using eight scene seeds with two repetitions per seed (256 episodes per category). For RoboTwin, the development matrix contains 50 tasks, each evaluated in 4 randomized scenes with 2 repetitions (400 episodes per evaluated condition).

Models. We use $\pi_{0.5}$ [42] for LIBERO-Pro, RLDX-1 [24] for RoboCasa365, and LingBot-VLA [50] for RoboTwin as the learned manipulation policies. The online planner uses GPT-5.6 Sol. The Meta-agent

Table 1 Success rate (%) on the eight LIBERO-Pro suites: the four LIBERO-T suites under instruction redirection (top block) and the four LIBERO-S suites under position perturbation (bottom block). H_0 is the initial harness and H_{VISTA} is the final selected harness. The baseline rows are reported on the LIBERO-T suites.

Method	Spatial	Object	Goal	LIBERO-10	Mean
<i>LIBERO-T suites (instruction redirection)</i>					
OpenVLA [25]	0.0	0.0	0.0	0.0	0.0
π_0 [3]	0.0	0.0	0.0	0.0	0.0
$\pi_{0.5}$ [42]	40.0	65.0	40.0	50.0	48.8
HarnessVLA [54]	79.0	79.0	62.0	62.0	70.4
Initial harness H_0	18.8	49.4	16.3	6.9	22.8
Evolved harness H_{VISTA}	98.1	100.0	98.8	89.4	96.6
Δ over H_0	+79.4	+50.6	+82.5	+82.5	+73.8
<i>LIBERO-S suites (position perturbation)</i>					
Initial harness H_0	30.0	14.4	4.4	0.0	12.2
Evolved harness H_{VISTA}	100.0	99.4	97.5	76.9	93.4
Δ over H_0	+70.0	+85.0	+93.1	+76.9	+81.3

Table 2 Success rate (%) on Composite Seen and Composite Unseen task categories in RoboCasa365. H_0 is initialized from HarnessVLA. Both categories report task-specific selected harnesses with complete evaluation matrices at the 2026-10-17 reporting cutoff.

Method	Composite Seen	Composite Unseen
$\pi_{0.5}$	6.8%	0.9%
RLDX-1	17.4%	6.8%
Initial harness H_0	16.4%	4.7%
Evolved harness H_{VISTA}	24.2%	13.3%
Δ over H_0	+7.8	+8.6

is implemented with Codex, using GPT-6 Astra in the LIBERO-Pro and RoboCasa runs and GPT-6.1 Sol in the RoboTwin run. Alongside the learned policies, the public capability interface exposes perception and native robot control; for RoboTwin this includes motion-planned end-effector movement, wrist rotation, and gripper control. RoboTwin episodes are capped at 100 planner turns and 10,000 environment steps. Following the settings in RLDX-1 on RoboCasa365 evaluation, each episode is capped at a per-task step horizon (300–2900 steps), which bounds per-rollout cost, prevents unbounded planner exploration, and makes large-scale harness evolution tractable under fixed compute.

Baselines. We compare our method with end-to-end VLA models, including OpenVLA [25], π_0 [3], $\pi_{0.5}$ [42], and RLDX-1 [24], and agentic embodied systems, including HarnessVLA [54]. For LIBERO-Pro, our method starts from a minimal harness H_0 exposing only the VLA and a small set of primitive tools. For RoboCasa365, we re-implement the HarnessVLA [54] harness on our own infrastructure as our initial harness, with prompts, tool schemas, and the per-task memory corpus taken from the reference implementation. RoboTwin starts from a port of the reference RPent harness, containing policy hooks, prompts, memory, and bindings to LingBot-VLA and native perception/control tools, with no additional composed skills at initialization. The matched before/after comparison is between each benchmark’s initial harness and its task-specific selected version. Other baseline rows are reference results; their source evaluation protocols are not assumed to match our development matrices.

4.2 Main Results

Overall Performance. We evaluate VISTA across three benchmarks and report both initial-harness comparisons and reference baseline results. As shown in Table 1, the initial harness achieves 22.8% mean success on

Table 3 Success rate (%) on RoboTwin. The development matrix contains 50 tasks, each evaluated in 4 randomized scenes with 2 repetitions (400 episodes per evaluated condition). H_0 is the initial harness and H_{VISTA} is the final selected harness.

Method	RoboTwin
Initial harness H_0	48.5
Evolved harness H_{VISTA}	74.0
Δ over H_0	+25.5

the four LIBERO-T suites, while the task-specific selected harnesses achieve 96.6%, a gain of 73.8 percentage points. On the four LIBERO-S suites, the mean success rate rises from 12.2% to 93.4%; aggregated across all eight LIBERO-Pro suites, the success rate improves from 17.5% (224/1280) to 95.0% (1216/1280). On the multi-stage household tasks of RoboCasa365, H_{VISTA} progressively improves the success rate from 16.4% to 24.2% on Composite Seen, and from 4.7% to 13.3% on Composite Unseen. Both improve over H_0 ; the table also includes RLDX-1 reference results [24]. On RoboTwin, the selected harness improves the aggregate success rate from 48.5% (194/400) to 74.0% (296/400) (Table 3).

Evolution over Development. Figure 3 visualizes how autonomous success changes over development rounds. Each curve aggregates the retained harness scores over the ten tasks of one LIBERO-T suite; an unchanged version carries its previous score forward. VISTA produces substantial improvements across all evaluated suites. For the relatively easier suites, only two to three development rounds are sufficient to raise success rates close to saturation, while the more challenging suites continue to improve over additional iterations.

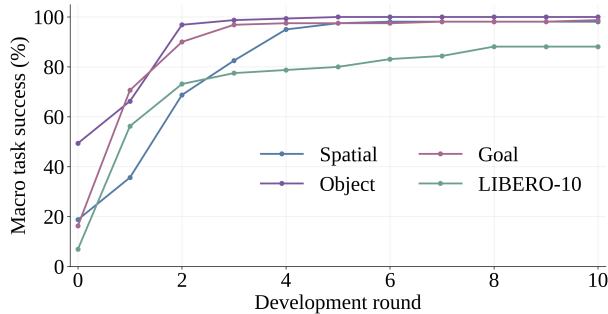


Figure 3 Success rates of the retained harnesses on the four LIBERO-T suites. Later points may reuse evaluations of unchanged selected versions.

Table 4 Recorded comparison on Goal-t00, t03, t05, and t06. Δ is the selected-harness gain over H_0 (percentage points); Token is mean cumulative Meta tokens per task. Actual token consumption is not matched between conditions.

Methods	R_1		R_2	
	Δ	Token	Δ	Token
Passive	+39.1	94.2M	+56.3	137.7M
Active	+50.0	42.2M	+85.9	91.6M

4.3 Ablation

Active vs. passive exploration. We compare VISTA with a rollout-only variant on Goal-t00, t03, t05, and t06. Both conditions permit history inspection, harness editing, and autonomous candidate evaluation. Only the active condition can open a live scene, inspect decision boundaries, and invoke diagnostic sensing or control capabilities. Each formal evaluation contains eight seeds with two repetitions.

The comparison uses the recorded runs, whose actual Meta token consumption differs. The first two passive rounds of Goal-t00 and Goal-t03 were conducted before the token-matching gate was introduced; later runs used token targets. Goal-t05 also used a different provider endpoint with the closest available reasoning-effort setting. These results therefore describe observed development outcomes, rather than a budget-matched estimate isolating tool access. Meta tokens include cached and uncached input, output, and sub-agent usage.

Across the four tasks, H_0 succeeds in 4/64 episodes. After the first round, the active and passive conditions reach 36/64 and 29/64; after the second, they reach 59/64 and 40/64. The corresponding gains over H_0 are reported in Table 4, alongside actual token consumption. Task-level effects vary: after two rounds, active versus passive success is 15/16 versus 0/16 on Goal-t00, 15/16 versus 14/16 on Goal-t03, 13/16 versus

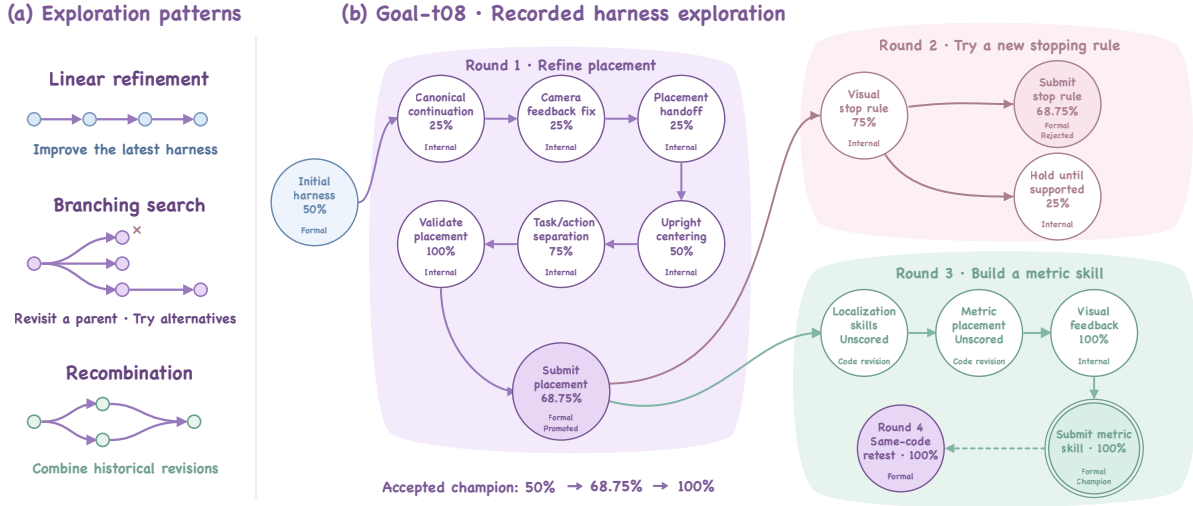


Figure 4 Harness development trajectories. (a) Representative development patterns observed during optimization, including linear refinement, branching exploration, and branch recombination. (b) A representative trajectory on LIBERO Goal-t08. Internal commit nodes denote small-scale active exploration and evaluation conducted by the Meta Agent during harness development, whereas **formal** commit nodes start evaluations on the complete 16-instance evaluation matrix and determine whether a candidate is promoted. Over the course of development, the harness improves from a 50% initial success rate to 100%.

16/16 on Goal-t05, and 16/16 versus 10/16 on Goal-t06. The aggregate difference is consistent with the value of additional interaction evidence, while the unequal resource usage and provider settings limit causal attribution.

4.4 Analysis: Inside Harness Evolution

Development Trajectories. During optimization, the Meta Agent maintains a Git-like lineage to explore and revise harnesses. We analyze the recorded lineages to characterize how harnesses evolve across successive revisions. Depending on the task, we observe different development patterns. In some cases, optimization proceeds linearly by refining the current harness. In others, the optimizer branches from an earlier revision when the current hypothesis appears ineffective and explores an alternative direction. We also observe cases where mechanisms developed in different branches are recombined into a new harness. Figure 4(b) shows a representative trajectory on LIBERO Goal-t08. Starting from an initial harness with 50% formal success, the first development round refines the placement behavior and promotes a candidate with 68.75% success. The next round explores an alternative stopping rule, but the submitted candidate does not improve over the incumbent and is therefore rejected. A later branch introduces a metric placement skill and reaches 100% formal success, after which the same frozen harness is re-evaluated and again obtains 100%. This trajectory illustrates that individual development attempts need not improve monotonically: VISTA can retain the best-so-far harness while continuing to explore alternative hypotheses and reuse evidence from unsuccessful branches.

Acquired Harness Mechanisms. We examine the artifacts produced during harness development to understand what is accumulated through optimization. The revisions span perception interfaces, state verification, action composition, and the allocation of control between executable skills and the frozen VLA. These categories are not mutually exclusive: a single revision may change both how an object is localized and how the outcome of the subsequent manipulation is verified. For example, across the developed harnesses, the optimizer introduces observation-dependent manipulation targets, explicit grasp and execution-state verification, bounded recovery procedures, and programmatic action sequences that complement VLA control.

Figure 5 shows a representative example from Goal-t09. The initial harness succeeds in 0/16 episodes. In

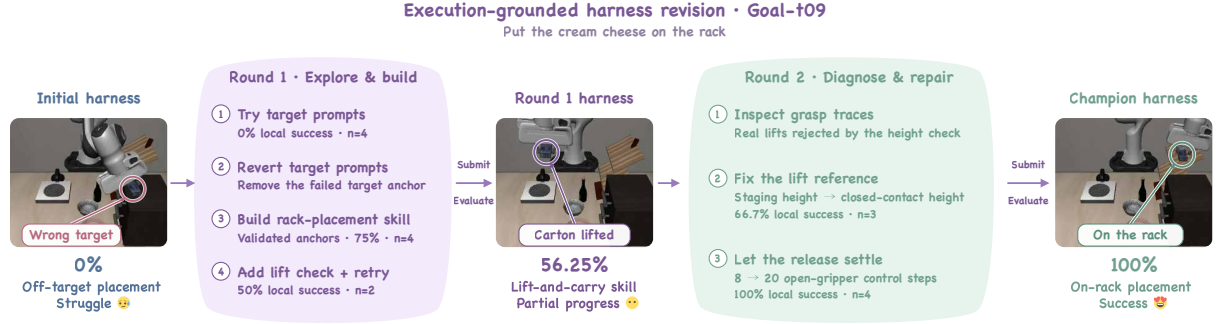


Figure 5 A representative harness development process on Goal-t09. The initial harness achieves 0/16 successes. The first development round adds lift verification, a bounded re-localization and grasp retry, and an explicit transport-and-release procedure, improving performance to 9/16. Analysis of the remaining failures then leads to a revision of the lift-state interpretation and release behavior, after which the final harness achieves 16/16 on the development set.

the first development round, the meta-agent analyzes the failed trajectories and makes several concrete changes: it requires a verified lift before transport, re-segments the carton before a bounded grasp retry, and introduces an explicit transport-and-release procedure for placing the carton onto the rack. The resulting harness reaches 9/16. The remaining failures expose a mismatch in the lift verification logic: lift was measured relative to an approach pose rather than the closed-contact state from which the carton was actually lifted. The second round therefore revises the detector to measure displacement over a continuous closed-contact segment and resets the accumulated evidence when contact is lost, together with further adjustments to the release behavior. After these revisions, the harness reaches 16/16 on the development set. The accepted changes from the first round remain part of the harness in the second round, so later development builds directly on mechanisms acquired earlier rather than restarting from the initial system.

5 Conclusion

We introduced VISTA, a framework for improving embodied agents by evolving their operational harness while keeping the underlying model parameters fixed. Rather than treating completed rollouts as a static source of feedback, VISTA enables a meta-agent to actively acquire diagnostic evidence, revise executable harness components, and test intermediate solutions before submitting a frozen candidate for autonomous evaluation. Across LIBERO-Pro, RoboCasa, and RoboTwin, this process progressively improves task success over successive development rounds. These results demonstrate that active, execution-grounded harness development provides a practical mechanism for converting embodied interaction into persistent improvements in agent behavior.

Limitations and Future Works. While VISTA demonstrates substantial gains across simulated benchmarks, our current evaluations are conducted strictly within high-fidelity simulation environments. Physical real-robot validation has not yet been performed. Real-world deployment inevitably introduces sensor noise, unpredictable actuation latency, and unmodeled contact dynamics. In future work, we plan to validate VISTA on physical robotic platforms (e.g., Franka Emika arms and mobile manipulators), and evaluate sensing, control constraints, and supervision needed for safe physical exploration.

References

- [1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.
- [2] Abrar Anwar, John Welsh, Joydeep Biswas, Soha Pouya, and Yan Chang. Remembr: Building and reasoning over long-horizon spatio-temporal memory for robot navigation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2838–2845. IEEE, 2025.

- [3] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Fusai, et al. π_0 : A vision-language-action flow model for general robot control. arXiv preprint arXiv:2410.24164, 2024.
- [4] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, et al. Rt-1: Robotics transformer for real-world control at scale. In arXiv preprint arXiv:2212.06817, 2022.
- [5] Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. Univla: Learning to act anywhere with task-centric latent actions. arXiv preprint arXiv:2505.06111, 2025.
- [6] Junting Chen, Yao Mu, Qiaojun Yu, Tianming Wei, Silang Wu, Zhecheng Yuan, Zhixuan Liang, Chao Yang, Kaipeng Zhang, Wenqi Shao, et al. Roboscript: Code generation for free-form manipulation tasks across real and simulation. arXiv preprint arXiv:2402.14623, 2024.
- [7] Siyi Chen, Hugo Hadfield, Alex Zook, Mikaela Angelina Uy, Chan Hee Song, Erwin Coumans, Xuning Yang, Faisal Ladhak, Qing Qu, Stan Birchfield, et al. Volo: A physical orchestrator for open-vocabulary long-horizon manipulation. arXiv preprint arXiv:2606.07723, 2026.
- [8] Tianxing Chen, Zhanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan-ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. arXiv preprint arXiv:2506.18088, 2025.
- [9] Yi Chen, Yuying Ge, Weiliang Tang, Yizhuo Li, Yixiao Ge, Mingyu Ding, Ying Shan, and Xihui Liu. Moto: Latent motion token as the bridging language for learning robot manipulation from videos. In 2025 IEEE/CVF International Conference on Computer Vision (ICCV), pages 19752–19763. IEEE, 2025.
- [10] Yongchao Chen, Jacob Arkin, Charles Dawson, Yang Zhang, Nicholas Roy, and Chuchu Fan. Autotamp: Autoregressive task and motion planning with llms as translators and checkers. In 2024 IEEE International conference on robotics and automation (ICRA), pages 6695–6702. IEEE, 2024.
- [11] Yongchao Chen, Yilun Hao, Yang Zhang, and Chuchu Fan. Code-as-symbolic-planner: Foundation model-based robot planning via symbolic code generation. In 2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 19248–19254. IEEE, 2025.
- [12] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. The International Journal of Robotics Research, 44(10-11):1684–1704, 2025.
- [13] Open X-Embodiment Collaboration, Abby O’Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, et al. Open X-Embodiment: Robotic learning datasets and RT-X models. <https://arxiv.org/abs/2310.08864>, 2023.
- [14] Ria Doshi, Homer Walke, Oier Mees, Sudeep Dasari, and Sergey Levine. Scaling cross-embodied learning: One policy for manipulation, navigation, locomotion and aviation. arXiv preprint arXiv:2408.11812, 2024.
- [15] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Vuong, et al. PaLM-e: An embodied multimodal language model. In Proceedings of the 40th International Conference on Machine Learning, volume 202 of Proceedings of Machine Learning Research, pages 8469–8488. PMLR, 2023.
- [16] Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, and Yijie Guo. Aha: A vision-language-model for detecting and reasoning over failures in robotic manipulation. arXiv preprint arXiv:2410.00371, 2024.
- [17] Jiafei Duan, Wentao Yuan, Wilbert Pumacay, Yi Ru Wang, Kiana Ehsani, Dieter Fox, and Ranjay Krishna. Manipulate-anything: Automating real-world robots using vision-language models. arXiv preprint arXiv:2406.18915, 2024.
- [18] Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, et al. Cap-x: A framework for benchmarking and improving coding agents for robot manipulation. arXiv preprint arXiv:2603.22435, 2026.
- [19] Liane Galanti, Dhruv Shah, and Tri Dao. Addressing the orchestration gap in generalist robots via physical agency. arXiv preprint arXiv:2607.21725, 2026.

- [20] Jiaheng Hu, Mohit Shridhar, Caden Lu, Dhruv Shah, Hao-Tien Lewis Chiang, Jie Tan, and Annie Xie. What matters in orchestrating robot policies: A systematic study of hierarchical vla agents. [arXiv preprint arXiv:2606.10267](#), 2026.
- [21] 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.
- [22] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. [arXiv preprint arXiv:2207.05608](#), 2022.
- [23] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner Monologue: Embodied reasoning through planning with language models. [arXiv preprint arXiv:2207.05608](#), 2022.
- [24] Dongyoung Kim, Huiwon Jang, Myungkyu Koo, Suhyeok Jang, Taeyoung Kim, et al. Rldx-1 technical report. [arXiv preprint arXiv:2605.03269](#), 2026.
- [25] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag 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. [arXiv preprint arXiv:2406.09246](#), 2024.
- [26] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pages 80–93. PMLR, 2023.
- [27] Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, et al. Causal world modeling for robot control. [arXiv preprint arXiv:2601.21998](#), 2026.
- [28] Ruiying Li, Yunlang Zhou, YuYao Zhu, Kylin Chen, Jingyuan Wang, Sukai Wang, Kongtao Hu, Minhui Yu, Bowen Jiang, Zhan Su, et al. Roboclaw: An agentic framework for scalable long-horizon robotic tasks. [arXiv preprint arXiv:2603.11558](#), 2026.
- [29] Siyao Li, Jiawei Gu, Shuai Liu, Kairui Hu, Zekun Li, Linjie Li, Chengcheng Tang, Po-Chen Wu, Ivan Shugurov, Lingni Ma, et al. Humanclaw: Can vision-language models act through a body? [arXiv e-prints](#), pages arXiv–2607, 2026.
- [30] Ziang Li, Dongzhou Cheng, Yibin Wang, Shiyue Wang, Xiaoyang Xu, Lingxuan Weng, Juan Wang, and Jiaqi Wang. Light-wam: Efficient world action models with state-fusion action decoding. [arXiv preprint arXiv:2606.08242](#), 2026.
- [31] 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, 2023.
- [32] Oscar Lima, Marc Vinci, Martin Günther, Marian Renz, Alexander Sung, Sebastian Stock, Johannes Brust, Lennart Niecksch, Zongyao Yi, Felix Igelbrink, et al. Agentic ai for robot control: Flexible but still fragile. In *Proceedings of the AAAI Symposium Series*, volume 8, pages 465–473, 2026.
- [33] Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19498–19508. IEEE, 2025.
- [34] Haowen Liu, Xirui Li, Shaoxiong Yao, Peng Shi, Tianyi Zhou, Jia-Bin Huang, Furong Huang, and Jiayuan Mao. Guava: An effective and universal harness for embodied manipulation. [arXiv preprint arXiv:2606.18363](#), 2026.
- [35] Jiangran Lyu, Kai Liu, Xuheng Zhang, Haoran Liao, Yusen Feng, Wenxuan Zhu, Tingrui Shen, Jiayi Chen, Jiazhao Zhang, Yifei Dong, et al. Lda-1b: Scaling latent dynamics action model via universal embodied data ingestion. [arXiv preprint arXiv:2602.12215](#), 2026.
- [36] Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *International Conference on Learning Representations (ICLR)*, 2026.

- [37] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.
- [38] Mingyu Ouyang, Siyuan Hu, Kevin Qinghong Lin, Hwee Tou Ng, and Mike Zheng Shou. Gameworld: Towards standardized and verifiable evaluation of multimodal game agents. In *European Conference on Computer Vision*, pages 41–59. Springer, 2026.
- [39] Abby O’Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024.
- [40] Bikang Pan, Fan Liu, Haotao Lu, Jingya Wang, and Ye Shi. Selfwam: A self-grounded unified world action model for fast robot control. *arXiv preprint arXiv:2608.00725*, 2026.
- [41] Jiaqi Peng, Xiqian Yu, Delin Feng, Yuqiang Yang, Wenzhe Cai, Jing Xiong, Ganlin Yang, Jinliang Zheng, Jiafei Cao, Xueyuan Wei, et al. Cortex: A bidirectionally aligned embodied agent framework for long-horizon manipulation. *arXiv preprint arXiv:2607.05377*, 2026.
- [42] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Galliker, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.
- [43] Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, et al. Spatialvla: Exploring spatial representations for visual-language-action model. *arXiv preprint arXiv:2501.15830*, 2025.
- [44] Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning. *arXiv preprint arXiv:2307.06135*, 2023.
- [45] Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, et al. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. *arXiv preprint arXiv:2502.19417*, 2025.
- [46] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.
- [47] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.
- [48] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.
- [49] Hongtao Wu, Ya Jing, Chilam Cheang, Guangzeng Chen, Jiafeng Xu, Xinghang Li, Minghuan Liu, Hang Li, and Tao Kong. Unleashing large-scale video generative pre-training for visual robot manipulation, 2023.
- [50] Wei Wu, Fan Lu, Yunnan Wang, Shuai Yang, Shi Liu, Fangjing Wang, Qian Zhu, He Sun, Yong Wang, Shuailei Ma, Yiyu Ren, Kejia Zhang, Hui Yu, Jingmei Zhao, Shuai Zhou, Zhenqi Qiu, Houlong Xiong, Ziyu Wang, Zechen Wang, Ran Cheng, Yong-Lu Li, Yongtao Huang, Xing Zhu, Yujun Shen, and Kecheng Zheng. A pragmatic VLA foundation model. *arXiv preprint arXiv:2601.18692*, 2026.
- [51] Kechun Xu, Zhenjie Zhu, Anzhe Chen, Rong Xiong, and Yue Wang. Apt: Action expert pretraining improves instruction generalization of vision-language-action policies. *arXiv preprint arXiv:2606.12366*, 2026.
- [52] John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.
- [53] Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

- [54] Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiayuan Liu, Wenhao Tang, Zhengru Fang, Yi Nie, Changxu Wei, Yu Wang, Wenbo Ding, and Chao Yu. Harness VLA: Steering frozen VLAs into reliable manipulation primitives via memory-guided agents. arXiv preprint arXiv:2607.08448, 2026. Version 4. Code: <https://github.com/RLinf/RPent>.
- [55] Yifan Zhong, Fengshuo Bai, Shaofei Cai, Xuchuan Huang, Zhang Chen, Xiaowei Zhang, Yuanfei Wang, Shaoyang Guo, Tianrui Guan, Ka Nam Lui, et al. A survey on vision-language-action models: An action tokenization perspective. arXiv preprint arXiv:2507.01925, 2025.
- [56] Jian Zhou, Xunyi Zhao, Gengze Zhou, Zerui Li, Sihao Lin, Jiajun Liu, and Qi Wu. Embodied agents take control: Minimal-interface zero-shot agents rival industrial-scale policies in vision-and-language navigation. arXiv preprint arXiv:2607.26148, 2026.
- [57] Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. LIBERO-PRO: Towards robust and fair evaluation of vision-language-action models beyond memorization. arXiv preprint arXiv:2510.03827, 2025.
- [58] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Vuong, et al. 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.

Appendix

A AI Assistance Disclosure

In this work, we used generative AI tools for assisting with translation, implementing partial code, supporting qualitative analysis of failure cases, and interpreting experimental results. We have not used generative AI tools for generating synthetic datasets, developing core theoretical or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proofs, assisting with proof writing, nor proposing or refining the main research hypothesis. Formulating mathematical proofs is not applicable to this work.

The authors are responsible for the final manuscript, reported results, and released artifacts, including content prepared with AI assistance. Harness candidates produced by the method are assessed through the autonomous evaluation procedure described in this paper.

B Additional Experimental Details

B.1 Infrastructure

VISTA separates harness development from autonomous evaluation through three interfaces. The meta-agent can read previous execution records, edit a candidate package, run live diagnostics, and request autonomous tests. The host freezes package versions, starts simulator processes, records traces, and applies the final selection rule. Figure A1 shows the three execution paths. Historical inspection reads completed records and does not recreate a previous simulator state. Live development starts a private scene from a frozen harness snapshot. Autonomous evaluation starts fresh episodes with a frozen candidate and no meta-agent intervention. All paths write records with explicit episode, request, and harness identities.

A live scene is not hot-swapped when the worktree changes. To test a later revision, VISTA starts a new session with the new frozen package. Live observations are stored as diagnostic evidence and are not silently added to the planner’s context. Historical depth is available only for trace modes that archive metric depth and calibration; `full_rgb` records contain RGB but no historical depth.

B.2 Meta-Agent Skill and Tool Surface

The meta-agent follows the `evolve-harness` skill. This skill describes how to inspect history, formulate a concrete hypothesis, revise the harness, use live sessions for diagnosis, and validate the resulting change. It is a development protocol; it does not change the planner or VLA weights. Table A1 summarizes the main interfaces exposed to the meta-agent.

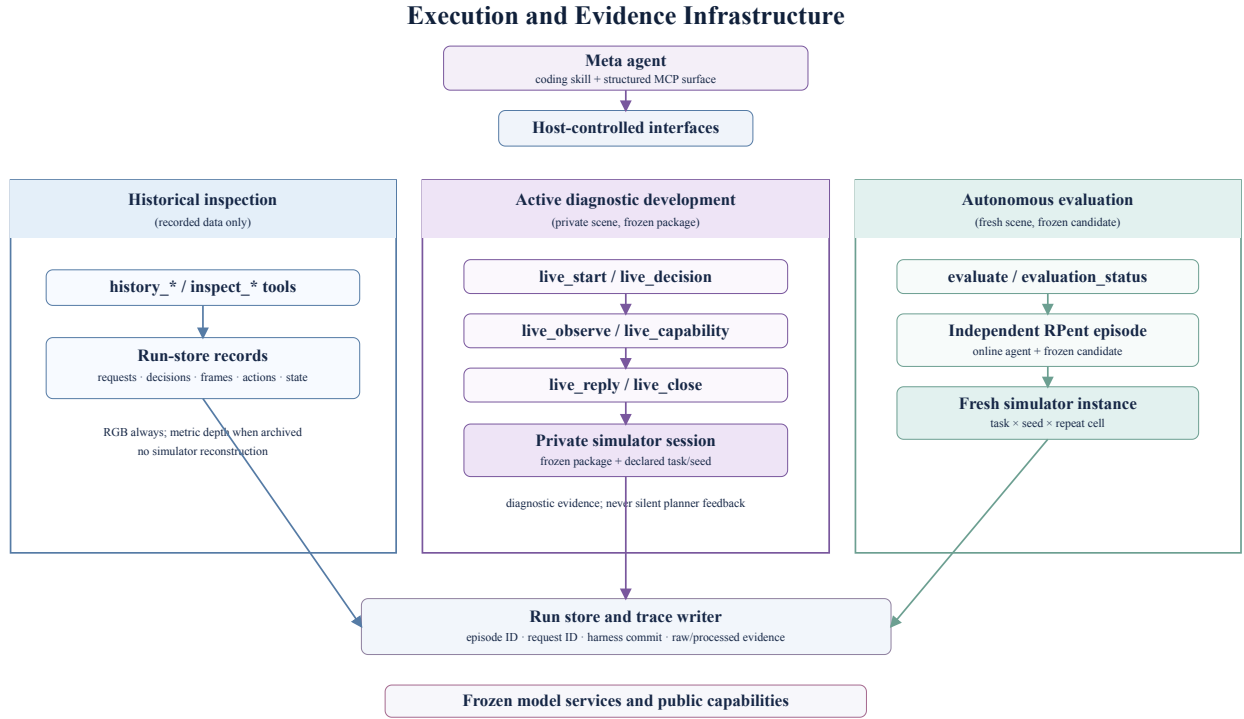


Figure A1 Execution and evidence infrastructure. The meta-agent communicates through a structured MCP interface. Historical inspection reads completed records; live development interacts with a private scene created from a frozen harness snapshot; and autonomous evaluation runs fresh episodes with no meta-agent intervention. All lanes use request, episode, and harness identities. Evidence is written to the run store, while package freezing and formal selection remain host-controlled. Historical depth is available only for trace modes that archive metric depth and calibration.

Table A1 Meta-agent interfaces in the VISTA implementation. Tools are grouped by their role in harness development.

Interface	Representative tools	Returned evidence or effect
History inspection	<code>history_summary</code> , <code>history_episodes</code> , <code>history_episode</code> , <code>history_decisions</code> , <code>history_decision</code> , <code>history_timeline</code> , <code>history_frame</code> , <code>inspect_decision</code> , <code>inspect_harness</code>	Versioned harness metadata, task/seed identities, exact planner decisions, tool arguments, raw and processed feedback, physical step ranges, images, and package contents.
Package development	Coding-agent file tools, <code>select_parent</code> , <code>budget</code> , <code>submit</code>	Candidate edits, saved drafts, selected Git parents, budget state, package validation, commit identity, and submission notes.
Active diagnostics	<code>live_start</code> , <code>live_decision</code> , <code>live_observe</code> , <code>live_capability</code> , <code>live_reply</code> , <code>live_close</code>	A private frozen-package session, decision-boundary observations, public capability probes, optional decision replacement, and diagnostic episode records.
Autonomous testing	<code>evaluate</code> , <code>evaluation_status</code>	A frozen candidate and fresh autonomous episodes for specified task–seed–repeat cells. Autonomous tests requested during development are diagnostic evidence only. Promotion is based on a separate host-initiated evaluation after submission.

The harness package itself exposes a separate runtime surface to the online agent. In the LIBERO reference package, this surface includes native file and termination tools, image reading, an observation skill, and a VLA-step skill. A package may additionally define a `Policy` with `prepare_request`, `process_tool_result`, and optional `decide` hooks. These hooks operate inside the ordinary host planner loop: they can transform the visible request or choose among bound tools, but they do not replace the host loop or change the model weights and budgets.

The distinction between the two surfaces is deliberate. Meta tools control development and inspection; harness tools determine what the online agent can use during an autonomous episode. A live call made by the meta-agent therefore does not imply that the corresponding operation was available to, or selected by, the online agent in the formal evaluation.

B.3 Git-Backed Harness History

Each frozen harness is stored as a Git commit. Ordinary revisions create parent edges, and an explicit merge creates a commit with multiple parents. The run store attaches evaluation requests, episode records, live diagnostics, and selection decisions to the corresponding commit.

Rejected candidates remain reachable through retained references and may be selected as later development bases. The incumbent used for formal comparison is recorded separately from the parent selected for revision. Thus, Git records code lineage, while the run store records the evidence and decisions associated with each version. Reusing a file or an idea does not create a Git ancestry edge.

C Additional Quantitative Results

C.1 Complete Per-Task Results

Tables A2 and A3 report complete per-task results on the development instances of the four LIBERO-T suites of LIBERO-Pro and RoboCasa, respectively. H0 denotes the unmodified harness, while Evolved denotes the retained champion. Both are evaluated on a 16-cell development matrix for each task.

LIBERO-Pro. Table A2 shows improvements over H0 on all 40 tasks across the four LIBERO-T suites. The suite-average success rate increases from 18.8% to 98.1% on Spatial, from 49.4% to 100.0% on Object, from 16.3% to 98.8% on Goal, and from 6.9% to 89.4% on LIBERO-10. The retained champions achieve 100.0% success in the formal development evaluation on 34 of the 40 tasks, including every task in Object. The task-level breakdown also reveals differences that are less apparent from the suite averages. Six tasks remain below 100% success: two in Spatial, one in Goal, and three in LIBERO-10. In particular, LIBERO-10 T4 and T9 remain at 62.5% and 43.8%, respectively, despite improving substantially over their zero-success initial harnesses.

Table A2 Per-task success rate (%) on the four LIBERO-T suites of LIBERO-Pro. $\pi_{0.5}$ denotes the direct-deploy baseline, H0 denotes the unmodified harness, and Evolved denotes the per-task champion after evolution. Δ is the gain over H0 in percentage points.

Suite	Method	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	Avg.
Spatial	$\pi_{0.5}$	0.0	100.0	0.0	0.0	0.0	50.0	100.0	50.0	0.0	100.0	40.0
	H0	0.0	93.8	0.0	0.0	0.0	56.3	18.8	0.0	12.5	6.3	18.8
	Evolved	100.0	100.0	100.0	100.0	93.8	100.0	100.0	87.5	100.0	100.0	98.1
	Δ	+100.0	+6.3	+100.0	+100.0	+93.8	+43.8	+81.3	+87.5	+87.5	+93.8	+79.4
Object	$\pi_{0.5}$	100.0	100.0	75.0	0.0	0.0	25.0	75.0	75.0	100.0	100.0	65.0
	H0	93.8	81.3	68.8	0.0	56.3	12.5	6.3	18.8	75.0	81.3	49.4
	Evolved	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	Δ	+6.3	+18.8	+31.3	+100.0	+43.8	+87.5	+93.8	+81.3	+25.0	+18.8	+50.6
Goal	$\pi_{0.5}$	0.0	100.0	25.0	25.0	100.0	0.0	75.0	50.0	25.0	0.0	40.0
	H0	0.0	6.3	31.3	0.0	31.3	0.0	25.0	18.8	50.0	0.0	16.3
	Evolved	100.0	100.0	87.5	100.0	100.0	100.0	100.0	100.0	100.0	100.0	98.8
	Δ	+100.0	+93.8	+56.3	+100.0	+68.8	+100.0	+75.0	+81.3	+50.0	+100.0	+82.5
LIBERO-10	$\pi_{0.5}$	25.0	75.0	100.0	0.0	0.0	75.0	100.0	50.0	75.0	0.0	50.0
	H0	0.0	25.0	37.5	0.0	0.0	0.0	6.3	0.0	0.0	0.0	6.9
	Evolved	100.0	100.0	100.0	87.5	62.5	100.0	100.0	100.0	100.0	43.8	89.4
	Δ	+100.0	+75.0	+62.5	+87.5	+62.5	+100.0	+93.8	+100.0	+100.0	+43.8	+82.5

RoboCasa. Table A3 reports the corresponding per-task results on Composite-Seen and Composite-Unseen. On Composite-Seen, the evolved harness increases aggregate success from 42/256 (16.4%) to 62/256 (24.2%), with 11 of the 16 tasks improving over H0 and the largest per-task gains on PackIdenticalLunches, PreSoakPan, and WashLettuce. On Composite-Unseen, the selected versions with complete evaluation matrices at the 2026-10-17 reporting cutoff improves from 12/256 (4.7%) to 34/256 (13.3%). Here as well, the gains are distributed across a broad set of tasks: 11 of the 16 tasks improve over H0, with the largest increases on CategorizeCondiments and BreadSelection.

Table A3 Complete per-task results on RoboCasa composite suites. The $\pi_{0.5}$ column reports the official RC365 evaluation (per-task success rate, %). H0 and Evolved report successes out of the 16-cell matrix. Both categories report task-specific selected harnesses with complete evaluation matrices at the 2026-10-17 reporting cutoff.

Composite-Seen					Composite-Unseen				
Task	$\pi_{0.5}$ (%)	H0	Evolved	Δ	Task	$\pi_{0.5}$ (%)	H0	Evolved	Δ
DeliverStraw	0	1/16	2/16	+1	ArrangeBreadBasket	2	2/16	3/16	+1
GetToastedBread	0	0/16	0/16	+0	ArrangeTea	0	0/16	0/16	+0
KettleBoiling	2	1/16	3/16	+2	BreadSelection	2	0/16	3/16	+3
LoadDishwasher	2	6/16	7/16	+1	CategorizeCondiments	0	0/16	6/16	+6
PackIdenticalLunches	4	0/16	3/16	+3	CuttingToolSelection	8	1/16	2/16	+1
PreSoakPan	26	4/16	7/16	+3	GarnishPancake	0	5/16	5/16	+0
PrepareCoffee	2	0/16	1/16	+1	GatherTableware	0	0/16	1/16	+1
RinseSinkBasin	4	6/16	7/16	+1	HeatKebabSandwich	0	0/16	0/16	+0
ScrubCuttingBoard	2	6/16	7/16	+1	MakeIceLemonade	0	1/16	1/16	+0
SearingMeat	0	0/16	0/16	+0	PanTransfer	0	0/16	1/16	+1
SetUpCuttingStation	10	2/16	2/16	+0	PortionHotDogs	0	0/16	1/16	+1
StackBowlsCabinet	26	6/16	8/16	+2	RecycleBottlesByType	0	0/16	2/16	+2
SteamInMicrowave	2	3/16	5/16	+2	SeparateFreezerRack	0	0/16	0/16	+0
StirVegetables	6	2/16	2/16	+0	WaffleReheat	0	2/16	4/16	+2
StoreLeftoversInBowl	0	1/16	1/16	+0	WashFruitColander	2	0/16	2/16	+2
WashLettuce	22	4/16	7/16	+3	WeighIngredients	0	1/16	3/16	+2
Avg / Total	6.8	42/256	62/256	+20	Avg / Total	0.9	12/256	34/256	+22

C.2 Complete Evolution Details

Figure A2 presents the recorded development trajectories for the four LIBERO-T suites. The heatmaps show the performance of the candidate evaluated in each round, whereas the curves below aggregate the best-so-far champion scores across tasks. The right-hand panels summarize the gain of each retained champion over H0. This distinction separates the quality of individual proposals from the performance accumulated through candidate selection.

Across the four suites, the largest aggregate improvements occur during the early rounds, followed by smaller, task-dependent gains. Object reaches perfect development performance, while Spatial and Goal approach saturation. LIBERO-10 improves more gradually and retains a larger performance gap at the reporting endpoint. The per-task trajectories further show that progress is not synchronized across tasks: some reach high success rates within the first few rounds, whereas others continue to receive revisions throughout the displayed development period. For example, Goal T1 reaches 100.0% only in R10.

Individual candidate scores are often non-monotonic, even when the retained champion remains unchanged. For example, Spatial T7 reaches 87.5% in R1, but several subsequent candidates obtain lower scores without replacing that champion. A plateau in the aggregate curve therefore does not necessarily indicate that development has stopped; it may instead reflect continued exploration without a successful replacement. Likewise, the non-decreasing champion curves follow from best-so-far selection and do not imply that every proposed revision improves performance.

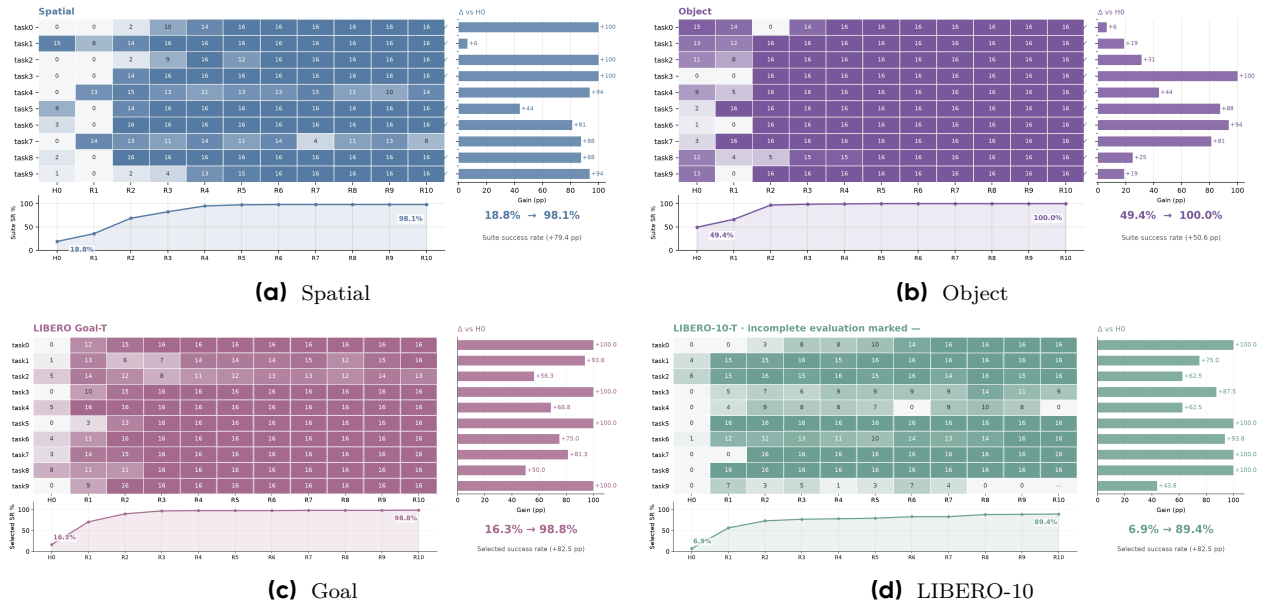


Figure A2 Per-task evolution on the four LIBERO-T suites. Rows are tasks and columns are development rounds (H0: initial harness). Solid cells report candidate successes out of 16 episodes. Dashed cells carry forward the last retained result after development stops; they do not denote new evaluations. An em dash marks an incomplete evaluation (LIBERO-10 task9, R10). Right: retained-harness gain over H0. Bottom: aggregate retained scores, which may reuse earlier evaluations.