Scaling by Structure: What Forms Can Scaling Take in Embodied AI?

Huawei Noah EAI Team

What forms can scaling take in embodied intelligence? Modern discussions of scaling have largely centered on increasing training data, neural network size, and computation. Yet an earlier tradition in symbolic AI pursued a complementary route: expanding the knowledge, rules, and procedures available to an intelligent system. These perspectives emphasize different forms of growth: model training on one side, and explicit, executable structure on the other. As multimodal large language models become increasingly capable of writing and revising code, code-as-policy offers a bridge between these traditions. This raises a central question: can embodied intelligence scale through the construction and refinement of symbolic and programmatic structure? Our BEHAVIOR-1K solution adopts hierarchical physical recursive self-improvement (HiPhys-RSI) to explore this direction. With the VLM backbone frozen, an agent builds robotic pipelines using classical perception, planning, and control methods, then diagnoses failures and revises its code using execution feedback and videos. Knowledge accumulates as reusable procedures, tools, constraints, and conditional behaviors. Inference-time routing selects among these pipelines and VLA policies. We compare this system with multiple vision-language-action models trained end to end, examining task performance, generalization, and feedback-driven refinement. We also use pipeline-generated trajectories to fine-tune VLAs, exploring if executable program refinement can support learned policy improvement.



1 Introduction

Scaling has become a central organizing principle in modern AI. In contemporary usage, it commonly refers to increasing the amount of training data, the size of neural networks, and the computation devoted to training (OpenAI, 2024; Liu et al., 2024; Yang et al., 2025; Team et al., 2023) and inference (Wei et al., 2022; Yao et al., 2023). The underlying ambition is to acquire increasingly general capabilities through learning at scale (Feigenbaum et al., 1977). An earlier tradition in symbolic AI emphasized a complementary form of growth: the accumulation and organization of structured knowledge (Feigenbaum, 1992). Intelligent behavior was supported by rules, operators, procedures, and other representations that a system could reason over and execute. From this perspective, capabilities could grow by expanding what the system knew, what operations it could perform, and how those operations could be composed (Newell and Simon, 2007). These traditions emphasize different dimensions of scaling: learned model capacity and structured, executable knowledge (McCarthy et al., 1960). General-purpose vision-language models (VLMs) connect these traditions through code-as-policy (Liang et al., 2023; Fu et al., 2026): they translate learned knowledge into executable skills and refine them using visual observations and execution feedback. This convergence raises a broader question: can intelligence scale through the construction and refinement of executable structure?

This question is particularly compelling in robotics, where a substantial repertoire of effective computational methods already exists. Geometric reasoning (Jiménez et al., 2001), task and motion planning (Garrett et al., 2021), SLAM (Cadena et al., 2016), perception (Mascaro and Chli, 2025), and feedback control (Tinoco et al., 2025) provide strong solutions within their respective operating domains. Applying these methods across diverse tasks and environments, however, often requires substantial human engineering and analysis. A pipeline that works in one setting may need a different grasp strategy, a revised action order, or an additional recovery procedure in another. The challenge therefore lies not only in the capabilities of individual modules, but also in how they are composed and adapted to task requirements. Coding agents offer a way to automate part of this work through agentic recursive self-improvement (RSI): they can examine execution feedback and

videos, diagnose failures, revise procedures, and test the resulting behavior. Across iterations, useful revisions can be retained as reusable tools and conditional behaviors, turning task-specific experience into structured knowledge that can support subsequent tasks. This perspective broadens what it means for a robotic system to generalize. Beyond executing a fixed policy in unfamiliar situations, a system may construct and adapt solutions using existing tools (Lu et al., 2026). If the resulting procedures can be reused across tasks and environments, capability growth may come from accumulating and refining this structured knowledge. This echoes the symbolic AI tradition of compiling problem-solving experience into reusable rules and operators that reduce the need for fresh reasoning in subsequent situations (Laird et al., 1986). As this knowledge becomes richer and better organized, a growing share of each new task can be handled by established structures, leaving progressively less to be constructed or revised. In this sense, executable knowledge scales through its expanding reach: accumulated structures support broader task coverage while requiring less additional adaptation. Our central question is whether automated programming, tool use, and feedback-driven refinement can sustain this form of scaling in embodied systems.

We investigate this question through our solution to the BEHAVIOR-1K challenge (Li et al., 2023). Its long-horizon household tasks require diverse interactions, coordinated action sequences, and adaptation to changing physical states, providing a demanding setting for studying how executable structure supports complex robot behavior. Our solution, hierarchical physical recursive self-improvement (HiPhys-RSI), uses a general-purpose coding agent to assemble robotic components into task pipelines and refine them through agentic RSI. Here, hierarchical improvement encompasses high-level tool use and coordination, refinement of low-level tool implementations, and VLA post-training using pipeline-generated trajectories. The resulting hybrid system supports inference-time routing among constructed pipelines and vision-language-action (VLA) policies. We compare this system with multiple VLA-only training variants, examining task performance and the effects of different training settings and input representations. Through this study, we explore scaling through structured knowledge acquired via RSI as a complementary path for embodied intelligence, where capabilities grow through the accumulation and refinement of executable procedures.

Our framework also connects executable program refinement with robotic foundation-model training. We combine trajectories generated by executable pipelines with benchmark demonstrations to fine-tune VLAs, incorporating experience acquired through pipeline construction and refinement into learned policies (Qi et al.). The resulting policies are available for inference-time routing alongside the constructed pipelines. This provides a basis for future co-evolution between executable structure and learned policies: improved policies could expand the capabilities available for subsequent pipeline construction, while further pipeline refinement could generate additional training experience. Exploring this reciprocal process is a promising direction for connecting scaling by structure with scaling by data.

2 Related Work

2.1 Established Domain-Specific Robotic Tools

Robotics has developed a broad repertoire of methods that address distinct computational problems and provide reusable building blocks for embodied systems (Jiménez et al., 2001; Mascaro and Chli, 2025; Garrett et al., 2021; Zhao et al., 2024; Tinoco et al., 2025). Geometric reasoning methods characterize spatial relationships and assess collision and clearance (Gilbert et al., 1988). Perception methods extract useful information from sensory observations through tasks such as local feature extraction, correspondence estimation, and point cloud alignment (Lowe, 2004; Besl and McKay, 1992). Task and motion planning methods connect discrete task decisions with continuous geometric feasibility through task decomposition, motion reasoning, and sampling of continuous parameters (Kaelbling and Lozano-Pérez, 2011; Garrett et al., 2020; LaValle, 1998). Trajectory optimization methods refine robot motions to satisfy task objectives and motion constraints through techniques such as gradient-based optimization and sequential convex optimization (Ratliff et al., 2009; Schulman et al., 2014; Vats and Mondal, 2020). Control methods translate desired task behaviors into robot commands and regulate execution through feedback, including coordinating manipulator motion and force at the task level (Khatib, 1987). Together, these approaches establish a substantial algorithmic foundation, although their use requires appropriate inputs, assumptions, and interfaces. Our work builds on this foundation by studying how a general-purpose coding agent can select, compose, and adapt robotic tools

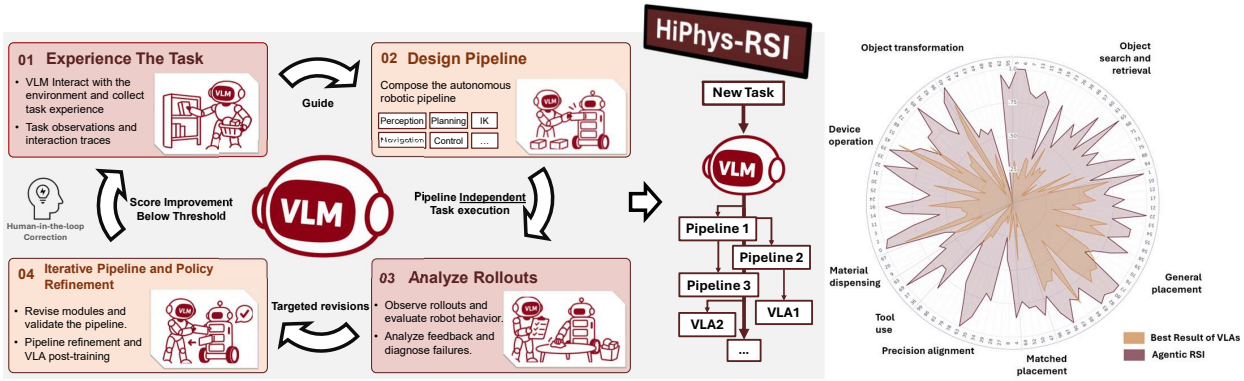


Figure 1 Overview of the proposed framework. HiPhys-RSI Workflow (Left): A coding agent, with a frozen VLM backbone, acquires task experience and iteratively refines executable robotic pipelines on train instances through tool composition, rollout analysis, and cross-task experience. Optional human corrections provide additional guidance during construction. **Inference-Time Routing Among Heterogeneous Pipelines and Policies (Middle):** The agent selects a pipeline or VLA policy for each new task using policy cards and the selected component executes autonomously. **Results (Right):** Task scores of our framework and the best-performing VLA baseline for each task, grouped by task category.

into executable task pipelines, automating the engineering needed to extend their applicability across tasks and environments.

2.2 Agentic Robotics

Language-model agents extend model capabilities through tool use, environment interaction, and feedback, including interleaved reasoning and actions, learned API invocation, and linguistic reflection (Yao et al., 2022; Schick et al., 2023; Shinn et al., 2023). Executable programs provide a unified action space and enable the accumulation of reusable skills (Wang et al., 2024, 2023). Software-engineering agents further demonstrate repository inspection, program editing, and execution-based debugging (Yang et al., 2024; Wang et al., 2025), providing a foundation for agent self-evolution through program and skill refinement (Liang et al., 2023; Fu et al., 2026). In robotics, agents incorporate environmental feedback into planning and generate programs that compose perception, planning, and control (Huang et al., 2022; Liang et al., 2023). Robotic harnesses support capability-aware coordination of heterogeneous policies through decomposition, routing, and memory-based bridging (Huang et al., 2026b), and combine frozen VLAs with analytic primitives and execution memories (Zhang et al., 2026b). Li et al. (2026b) expose geometric perception through queryable tools and distill execution traces into smaller models, connecting tool-mediated interaction with policy learning. Recent work extends these agentic methods toward robot self-evolution. Li et al. (2026a) integrate data collection, policy learning, and execution, while Xiao et al. (2026) use coding agents to refine policies and training infrastructure through physical trials. Wen et al. (2026) organize skills hierarchically, localize failures, and validate revisions before reuse. Liang et al. (2026) treat the supporting system as an evolving policy, converting execution traces into validated updates to context management, persistent memory, and hierarchical skills. Zhang et al. (2026a) co-evolve orchestrators and policies through deployment-derived demonstrations and verified policy updates. These studies explore capability growth through program revision, skill accumulation, system evolution, and policy learning, but do not systematically examine the scalability of structured tool use or comprehensively compare it with end-to-end approaches such as VLAs. Our work addresses this gap by studying how a general-purpose coding agent constructs and refines executable pipelines from established robotic tools, and systematically comparing these pipelines with learned action models.

3 Problem Formulation

We study how general-purpose coding agents construct autonomous robotic pipelines from task specifications, demonstrations, and reusable robotic tools. A pipeline construction problem is defined by $\mathcal{I} = (\tau, \mathcal{D}_\tau, \mathcal{E}, \mathcal{T})$, where τ specifies the task and its success conditions, $\mathcal{D}_\tau$ is a set of reference demonstrations, $\mathcal{E}$ is the robot environment, and $\mathcal{T}$ is a library of callable tools implementing classical methods for navigation, perception,

planning, and control. Previously constructed pipelines may also be included as reusable callable tools. These tools may differ in their inputs, outputs, and temporal scopes. The environment provides observations $o_t \in \mathcal{O}$ and accepts robot actions $a_t \in \mathcal{A}$. Reference demonstrations consist of previously collected videos and trajectories of a robot completing the corresponding task. For each task τ, let $\mathcal{C}_\tau^{\text{train}}$ denote the set of initial environment configurations used for task experience acquisition and pipeline refinement. Given $\mathcal{I}$, a frozen agent $\mathcal{C}_\theta$, with a frozen VLM backbone $\mathcal{V}_\phi$ for interpreting visual evidence, constructs an executable pipeline p. The pipeline defines tool composition, information flow, and conditional execution based on observations and tool outcomes. The resulting pipeline p autonomously executes the task by coordinating tool calls and responding to environmental feedback. Its execution is denoted by $\xi = \text{Rollout}(p, \mathcal{E})$, where ξ records the execution trajectory and task outcome. At inference time, the agent uses accumulated capability information to select an existing pipeline or VLA policy to accomplish a new task.

Pipeline construction proceeds through iterative execution and refinement on $\mathcal{C}_\tau^{\text{train}}$. At iteration k, the pipeline is executed to produce $\xi_k = \text{Rollout}(p_k, \mathcal{E}, c_k)$, where $c_k \in \mathcal{C}_\tau^{\text{train}}$ specifies the initial environment configuration. The resulting videos, execution logs, tool outputs, and task outcomes inform the next revision: $p_{k+1} = \mathcal{C}_\theta\big(\mathcal{I}, p_k, \mathcal{F}_\phi(\xi_k)\big)$, where $\mathcal{F}_\phi$ combines execution feedback with visual analysis from the frozen VLM $\mathcal{V}_\phi$. The parameters of the coding agent and VLM remain fixed throughout this process; improvements are encoded in the pipeline's executable procedures and associated structured knowledge. The objective is to construct a pipeline that maximizes the expected task score:

$$p^\star \in \arg\max_{p \in \mathcal{P}(\mathcal{T})} \mathbb{E}[G_\tau(\xi)]$$

Here, $\mathcal{P}(\mathcal{T})$ denotes the set of executable pipelines built using the available tools, and $G_\tau(\xi)$ measures task performance for execution ξ. The expectation accounts for variations in environment configurations and execution outcomes. During construction, performance is estimated on the fixed train-instance set $\mathcal{C}_\tau^{\text{train}}$ to guide refinement. Pipeline construction is subject to a given computation and interaction budget.

4 Method

Our framework enables a frozen VLM-based coding agent to construct autonomous robotic pipelines through four stages: task experience acquisition, pipeline design, rollout analysis, and iterative refinement. The agent grounds its design in demonstrations and interactions on the task's train-instance set, composes available robotic tools into executable procedures, and improves these procedures using execution feedback. The frozen VLM backbone supports the interpretation of visual evidence throughout this process. Previously constructed pipelines and accumulated task experience can subsequently be reused to support new tasks.

4.1 Task Experience Acquisition

For each task, we sample 10 instances from the BEHAVIOR-1K training environments to form $\mathcal{C}_\tau^{\text{train}}$. This set remains fixed throughout experience acquisition, rollout analysis, and pipeline refinement. The VLM-backed coding agent first examines the task specification τ and reference demonstrations $\mathcal{D}_\tau$, then directly controls tool invocation and robot interaction on a small number of instances from this set. During this exploratory stage, the agent interprets observations and tool outcomes, selects subsequent operations, and attempts to complete the task. These interactions provide firsthand evidence of which tools are needed, how they should be combined, and what observable conditions indicate successful completion. The resulting experience captures tool inputs and outputs, dependencies between operations, and intermediate states relevant to task progression. Together with the reference demonstrations, it helps the agent ground the specified success conditions in observable outcomes and identify where perception, coordination, or feedback is required. The agent then translates the tool-use decisions explored on individual train instances into executable and structured procedures, constructing an autonomous pipeline from the collected interaction traces.

4.2 Executable Pipeline Design

Building on the task experience acquired from $\mathcal{C}_\tau^{\text{train}}$, the coding agent constructs an initial pipeline p_0 by composing tools from $\mathcal{T}$ into an executable program. The agent translates the operations and dependencies

identified during exploration into tool calls, information flow, and conditional execution logic. To coordinate tools with different interfaces and temporal scopes, it implements the supporting code needed to prepare inputs, interpret outputs, maintain execution state, and transfer control between modules. Observable success conditions identified during exploration inform checks for intermediate progress and subtask completion. The resulting pipeline combines classical robotic modules within a single procedure. For example, perception outputs may parameterize a planner, whose resulting trajectory is executed by a controller. The program specifies when each component is invoked and how execution proceeds based on observations and tool outcomes. Its structure is thus informed by the agent's task experience and the capabilities of the available tools (Huang et al., 2026b). Once constructed, the pipeline executes autonomously according to its programmed logic, carrying out tool coordination that the agent previously performed directly during exploration. Within each rollout, the pipeline responds to environmental feedback through its implemented checks and conditional branches. During construction, the coding agent subsequently analyzes rollouts on the train-instance set and revises the program between executions, separating agent-driven pipeline construction from autonomous robot execution.

4.3 Rollout Analysis and Failure Diagnosis

At iteration k, the pipeline is executed under an initial configuration $c_k \in \mathcal{C}_\tau^{\text{train}}$ to obtain ξ_k. The resulting videos, execution logs, tool outputs, and task outcomes are combined into the feedback representation $\mathcal{F}_\phi(\xi_k)$. Program-level feedback reveals which operations were invoked and what they returned, while VLM-based visual analysis assesses the corresponding robot behavior and changes in the environment. These complementary signals help distinguish successful tool invocation from successful physical task execution. The agent compares the observed rollout with the intended task progression to identify where execution diverged and which pipeline components require revision. Failures may arise from incorrect tool inputs, incompatible assumptions between modules, inappropriate execution order, or insufficient handling of unsuccessful actions. Visual evidence also helps diagnose failures whose causes are not apparent from execution logs, such as a manipulation that finishes without achieving the intended object state or a motion planner that fails to find a path because of small obstacles. The purpose of this analysis is to connect observed failures to actionable program changes. Rather than using the task score alone, the agent uses the available execution evidence to determine what should be revised and why.

4.4 Iterative Pipeline Refinement

Conditioned on the task specification, current pipeline, and rollout feedback, the coding agent produces the next revision:

$$p_{k+1} = \mathcal{C}_\theta\big(\mathcal{I}, p_k, \mathcal{F}_\phi(\xi_k)\big).$$

Revisions may change tool selection, input and output transformations, execution order, tool configuration parameters, or conditional logic. The updated pipeline is then evaluated to determine whether these changes improve task performance. Repeated execution and analysis allow the agent to refine the coordination of existing capabilities while keeping the coding agent and VLM fixed.

Let s_k denote the mean evaluation score of pipeline p_k over $\mathcal{C}_\tau^{\text{train}}$. We measure the improvement after a revision as

$$\Delta s_k = s_{k+1} - s_k.$$

The refinement process terminates when the score improvement Δs_k remains below a threshold ϵ for a specified number of consecutive revisions, or when the prescribed computation or interaction budget is exhausted. The improvement threshold and patience window are task-dependent hyperparameters, with default values of $\epsilon = 0.02$ and 10 revisions, respectively, adjusted according to task length and difficulty. During construction and refinement, optional human-in-the-loop corrections can provide additional guidance for pipeline design and revision. Once constructed, the pipeline executes autonomously according to its programmed logic.

4.5 Cross-Task Experience and Pipeline Reuse

The construction process on each task's train-instance set produces both executable procedures and structured knowledge that can support subsequent tasks. Executable procedures preserve implemented tool compositions,

while structured knowledge captures relevant operating conditions, observed failure modes, and lessons from previous revisions. These resources allow the coding agent to build on prior experience when designing a new pipeline. For a new task, the agent can select and adapt previously constructed pipelines, incorporating them as callable components alongside the original robotic tools. This supports hierarchical composition: a pipeline may coordinate several reusable procedures, each of which invokes lower-level tools. Reused components are evaluated on the new task's train-instance set, and execution feedback from this set guides any necessary adaptation. Throughout this process, the coding agent and supporting VLM remain frozen. Improvements are encoded in the pipeline's executable logic and associated structured knowledge, allowing accumulated experience to improve how existing robotic capabilities are coordinated across tasks.

4.6 Tuning VLAs with Agent-Generated Trajectories

Agent-generated trajectories provide a pathway for transferring experience acquired through HiPhys-RSI into robotic foundation models. As executable pipelines are refined through interaction, their rollouts capture manipulation strategies, coordinated action sequences, and responses to execution failures (Qi et al.). We convert these rollouts into observation–action sequences paired with task instructions and combine them with benchmark demonstrations for task-specific VLA post-training, as detailed in Section 5. The resulting policies are included among the candidates for inference-time routing alongside the constructed pipelines. This connects scaling by structure with scaling by data: strategies and recovery behaviors developed through executable program refinement generate observation–action trajectories that complement the original demonstrations. These trajectories expose VLAs to alternative ways of approaching tasks and handling execution difficulties, providing a route for incorporating experience accumulated in explicit procedures into learned policy parameters.

4.7 Dynamic Routing at Inference Time

To support inference-time routing (Figure 1), we associate each available VLA policy and constructed pipeline with a *policy card*. Each card summarizes the component's capabilities and requirements, together with its historical success distribution across task contexts and execution conditions on the train-instance sets. For constructed pipelines, the card additionally incorporates corner-case documentation collected during construction and refinement on these sets, including observed failure modes and conditions under which execution becomes unreliable (Huang et al., 2026b). These cards provide the coding agent with evidence about both the applicability and limitations of each component. For each incoming task instance, the agent compares its context and requirements with the policy cards to select an appropriate pipeline or VLA policy. Routing accounts for the component's historical performance under relevant conditions and documented corner cases, alongside its functional suitability for the task. The selected component then executes the task according to its own execution mechanism, with tool coordination within a pipeline governed by its programmed logic. *Policy cards* are dynamically updated with new execution outcomes and observed corner cases from the train-instance sets, allowing subsequent routing decisions to incorporate accumulated experience.

5 Vision-Language-Action Models: Architecture and Training Settings

We adopt the first-place solution of the 2025 BEHAVIOR Challenge as our base model (Larchenko et al., 2025). This solution builds on pretrained $\pi_{0.5}$ weights (Intelligence et al., 2025) and adapts the model to household mobile manipulation through continued training on BEHAVIOR demonstrations. Its principal modifications include correlated noise for flow matching, learnable mixed-layer attention between the visual backbone and action expert, and task-stage prediction for conditioning action execution. We adapt this base model to the 100 tasks in the current benchmark and evaluate three training variants. Here, *meta-training* refers to joint training on the pooled benchmark demonstrations from all 100 tasks.

Meta-1-Epoch. We jointly train the base model on the pooled demonstration dataset covering all 100 tasks for one complete epoch. Training uses the full set of available benchmark training demonstrations for each task, allowing a single shared policy to learn navigation and manipulation behaviors across tasks. The resulting checkpoint is evaluated directly on all tasks and serves as the common initialization for subsequent task-specific post-training.

Per-Task Post-Training. We initialize each task-specific policy from the same Meta-1-Epoch checkpoint and continue training on data for the corresponding task. This dataset combines training demonstrations provided by the BEHAVIOR-1K benchmark with additional trajectories generated by agentic pipelines, enabling the transfer of behaviors developed through pipeline refinement into learned policies, as described in Section 4.6. This produces a separate policy for each task, retaining the initialization acquired through joint training while adapting to task-specific execution patterns and manipulation requirements. At evaluation, we select the policy corresponding to the target task. We additionally incorporate 3D spatial information, such as depth and robot-relative geometry, into policy training (Liu et al., 2026).

Meta-2-Epochs. We extend joint training on the same pooled benchmark demonstration dataset to two epochs in total. This variant evaluates the effect of additional multi-task training without task-specific specialization or augmentation with pipeline-generated trajectories.

Together, these VLA variants provide learned-policy candidates for inference-time routing alongside the constructed pipelines. They also serve as VLA-only comparison baselines for examining the effects of extended multi-task training and task-specific post-training. The per-task post-training setting additionally connects pipeline refinement with learned-policy improvement by incorporating agent-generated experience into VLA training.

6 Results and Discussions

6.1 Overall Performance Compared to the VLA Baselines

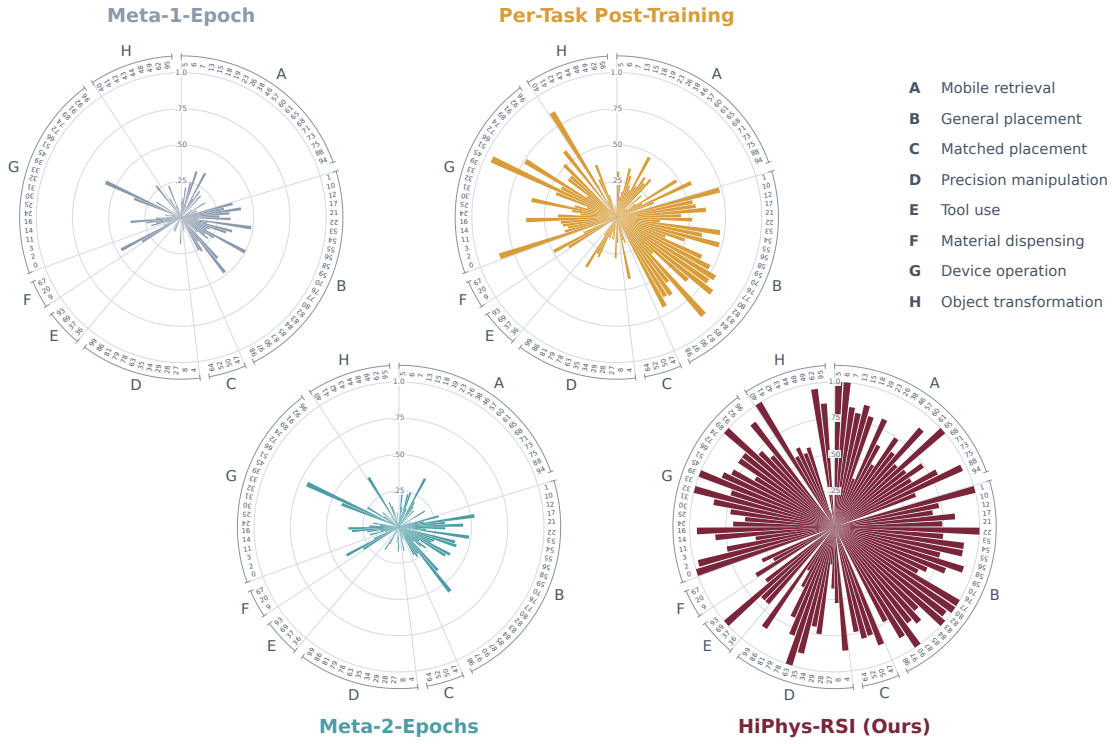


Figure 2 Per-task performance across eight task categories. We compare three VLA training variants with our final solution. Each radial bar represents the score for one task, with task IDs shown along the perimeter and tasks grouped into categories A–H as indicated in the legend. Detailed category definitions and categorization criteria are provided in Section A.

Coding agents unlock the scaling of structured knowledge. Our agentic system achieves higher scores than the VLA-only variants on nearly all tasks, with improvements spanning all eight task categories (Figure 2). These broad improvements are consistent with the benefits of combining explicit information processing, task-level reasoning, and specialized action execution within a hybrid system. Each stage serves a distinct role: classical

perception and SLAM modules transform observations into structured scene information; during pipeline construction and refinement, the VLM-based agent uses this information to devise execution strategies and specify goals and constraints that are encoded in the pipeline; specialized downstream executors translate these decisions into robot motions through their respective planning and control methods. For example, affordance-based path planning supports object search and navigation, feedback control enables precise alignment, task and motion planning handles logically structured sorting, and code-as-policy routines perform repeated pick-and-place operations. This division connects scene understanding to task-level decisions and then to physical execution, allowing complementary components to work together through explicit information flow. Realizing these benefits requires selecting suitable capabilities, coordinating their execution, and refining their implementations through environmental feedback. Scaling the system therefore depends on performing this engineering work efficiently across tasks. These processes traditionally demand substantial manual engineering, limiting the scalability of systems built on structured robotics knowledge. A general-purpose VLM-based coding agent automates much of this work through the RSI loop, translating execution feedback into revised implementations, constraints, and recovery behaviors. By reducing this engineering bottleneck, the agent provides a path to scaling the use of structured knowledge through the construction and refinement of executable pipelines.

Additional VLA training improves task scores but leaves persistent capability gaps. Figure 2 shows that extending joint meta-training from one epoch to two yields comparatively limited gains under the evaluated training configuration. In contrast, per-task post-training improves individual task scores through additional task-specific training on a mixture of benchmark demonstrations and pipeline-generated trajectories, while largely preserving the relative distribution of strengths and weaknesses across task categories. These gains reflect the combined effects of task-specific post-training and data augmentation. Across the evaluated VLA variants, general placement and selected object transformation tasks are relatively stronger, whereas mobile retrieval, matched placement, and precision manipulation remain challenging; device operation also shows uneven performance. We hypothesize that these persistent gaps partly reflect limitations in the evaluated models' long-horizon coordination and task-specific manipulation capabilities. In particular, the weaker performance on mobile retrieval may stem from difficulties in supporting extended decision-making for mapping, search, and exploration. The challenges in matched placement and precision manipulation may reflect limitations in feedback control for accurate object alignment. Uneven performance on device operation may likewise arise from difficulties in understanding physical mechanisms and device functions in context. These demands may help explain why additional training improves performance on some tasks while leaving the broader pattern of category-level weaknesses largely unchanged.

6.2 Structured Knowledge Scaling

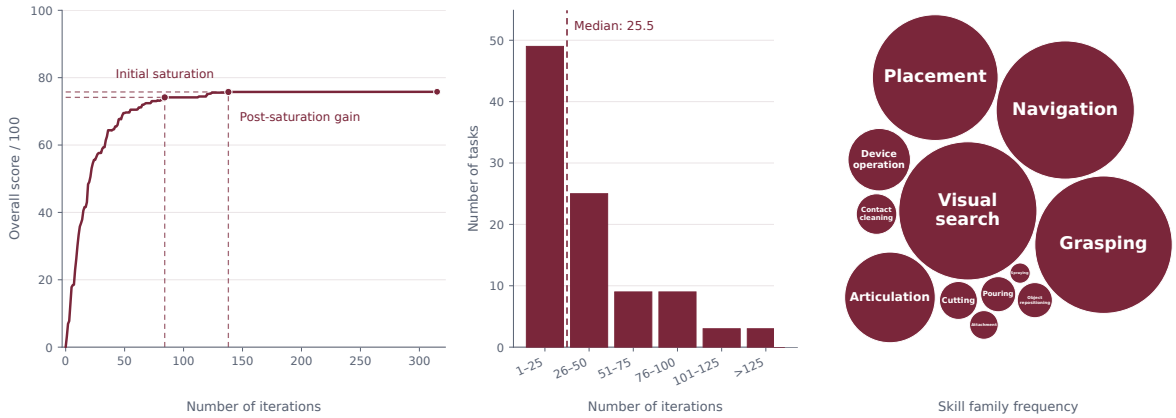


Figure 3 Pipeline refinement progress and skill composition. Left: Overall task score as a function of refinement iterations, showing rapid initial improvement followed by saturation and modest subsequent gains. **Middle:** Distribution of iteration counts across tasks. **Right:** Frequency of skill usage in the constructed pipelines, with bubble area proportional to frequency.

The trend of scaling structured knowledge. The refinement curve in Figure 3 (left) suggests that scaling

structured knowledge involves an initial phase of capability composition followed by a more demanding phase of strategy discovery. Early iterations yield rapid gains because many straightforward failures can be addressed by correcting tool configurations, execution order, and coordination within an otherwise suitable strategy. As these issues are resolved across tasks, aggregate performance gains diminish, with further improvements concentrated in the remaining difficult cases. The subsequent gains arise when continued exploration through RSI identifies more stable manipulation strategies absent from the provided training data or demonstration examples. These discoveries change how the task is approached, allowing the system to overcome failures that persist under the original strategy. The initial plateau therefore need not represent a fixed capability ceiling: it can mark the point at which further progress requires exploring alternative solutions. Improvements are retained in executable code, enabling knowledge acquired through interaction to accumulate while the underlying models remain frozen.

Broad coverage emerges early; remaining tasks demand disproportionate effort. The overall iteration distribution in Figure 3 (middle) is concentrated at low counts with a long right tail, suggesting that pipeline construction is relatively economical for most tasks but substantially more demanding for a small subset. This pattern highlights a scaling trade-off: extending coverage to increasingly difficult tasks can raise the refinement cost much faster than the number of tasks addressed. A uniform iteration budget would therefore risk spending unnecessary effort on tasks that stabilize early while prematurely stopping those that need further exploration. Adaptive allocation based on observed progress could better balance broad coverage with the cost of resolving the remaining difficult cases.

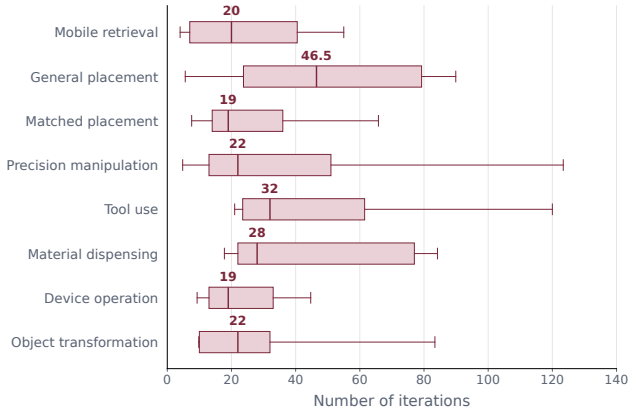


Figure 4 Pipeline refinement iterations by task category. Boxes show the 25th–75th percentiles, lines and labels indicate medians, and whiskers mark the 10th–90th percentiles.

Shared skills support breadth; specialized skills extend coverage. The skill involvement frequencies in Figure 3 (right) reveal why improvements to some components can have broad effects while others remain essential for extending coverage. Navigation, visual search, grasping, and placement recur because diverse household tasks share the need to locate, reach, acquire, and reposition objects. Their repeated use also creates multiple opportunities for errors to accumulate, making their reliability consequential for long-horizon success. Specialized skills such as articulation, device operation, cutting, and pouring occur less frequently, but can determine whether a particular task is achievable at all. Scaling therefore requires both improving shared capabilities and resolving specialized bottlenecks.

Refinement demands vary across task categories. The category-level distributions in Figure 4 reveal distinct sources of refinement effort. General placement requires the most refinement on average, reflecting the diversity of object geometries, support surfaces, and release conditions that must be accommodated by otherwise familiar pick-and-place routines. Tool use also demands substantial refinement, as successful execution requires coordinating the tool's grasp, orientation, and motion with the intended contact effect. Precision manipulation exhibits a particularly long upper tail despite a relatively low median: existing routines suffice for some tasks, while tight alignment tolerances and contact constraints make others considerably harder to refine. Material dispensing likewise shows substantial variation due to differences in container geometry and the motion needed to control material transfer. Device operation has a comparatively compact distribution and lower typical iteration counts because its interactions can be implemented through bounded, repeatable motions once the relevant mechanism is identified. These patterns show that refinement demands depend on the diversity and sensitivity of physical interactions within each category.

6.3 Generalization to Hidden Test

The hidden evaluation tests whether the constructed pipelines generalize beyond the instances used for development and reporting. As Figure 5 shows, most tasks exhibit lower Q scores on the hidden evaluation,

although the majority of these decreases remain below 0.2. Several tasks retain comparable performance or achieve higher scores, suggesting that the constructed procedures transfer across instances. The largest declines primarily occur in tasks that depend heavily on object search, where success is strongly affected by randomized initial object locations. Objects placed in occluded or difficult-to-access locations can substantially increase search time and prevent completion within the execution budget. Some tasks also have very few scoring predicates, with certain tasks using binary scores, so a single unmet condition can produce a disproportionately large score decrease. With only 10 test instances per task, the reported differences are also subject to sampling variability. The observed gaps therefore reflect sensitivity to instance configurations, coarse scoring granularity, and variability from the limited evaluation sample. Overall, the results indicate generalization across instances.

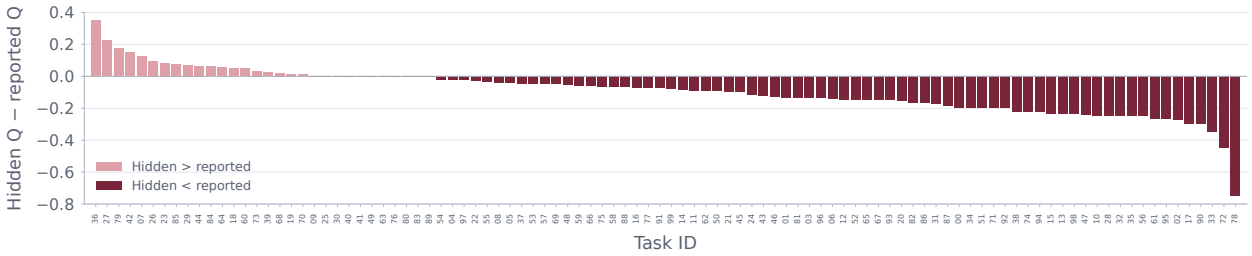


Figure 5 Per-task differences between hidden-evaluation and reported Q scores. Bars show $Q_{\text{hidden}} - Q_{\text{reported}}$, sorted in descending order. Positive values indicate higher scores on the hidden evaluation, while negative values indicate lower scores. Task IDs are shown along the horizontal axis.

The hidden evaluation tests whether the constructed pipelines generalize beyond the instances used for development and reporting. As Figure 5 shows, most tasks exhibit lower Q scores on the hidden evaluation, although the majority of these decreases remain below 0.2. Several tasks retain comparable performance or achieve higher scores, suggesting that the constructed procedures transfer across instances. The largest declines primarily occur in tasks that depend heavily on object search, where success is strongly affected by randomized initial object locations. Objects placed in occluded or difficult-to-access locations can substantially increase search time and prevent completion within the execution budget. Some tasks also have very few scoring predicates, with certain tasks using binary scores, so a single unmet condition can produce a disproportionately large score decrease. With only 10 test instances per task, the reported differences are also subject to sampling variability. The observed gaps therefore reflect sensitivity to instance configurations, coarse scoring granularity, and variability from the limited evaluation sample. Overall, the results indicate generalization across instances.

6.4 Failure Mode Analysis

Failures shift from manipulation errors toward execution bottlenecks. The failure distribution in Figure 6 shifts from a strong concentration in grasping and placement for the VLA baseline toward grasping and timeout in our approach. Placement, articulation, and object loss become less prominent, consistent with explicit geometric constraints and specialized execution routines addressing these interactions more reliably. Grasping remains the leading failure mode for both methods, highlighting the persistent difficulty of establishing stable contact across variations in object geometry, accessibility, and surrounding clutter.

Physical constraints limit further refinement. The prominence of timeout and navigation failures highlights the cost of locating and reaching target objects. These failures frequently co-occur when the robot cannot find a target object or path planning repeatedly fails in cluttered environments. Prolonged search, repeated planning attempts, and inefficient movement consume the execution budget, leaving insufficient time for subsequent manipulation. These failures reflect limitations of the current perception, search, and motion-planning strategies under constrained visibility, physical clearance, and execution budgets. A target object may be tucked behind larger objects in a cabinet corner, remaining invisible from accessible viewpoints despite repeated searches. Even after the target is located, reaching it may require navigating between closely spaced furniture or moving the arm through a narrow gap, where small errors in scene reconstruction or robot positioning make an otherwise feasible path collide with the surroundings. In such cases, changing the search order or repeatedly adjusting the planned trajectory does little to improve visibility or increase physical

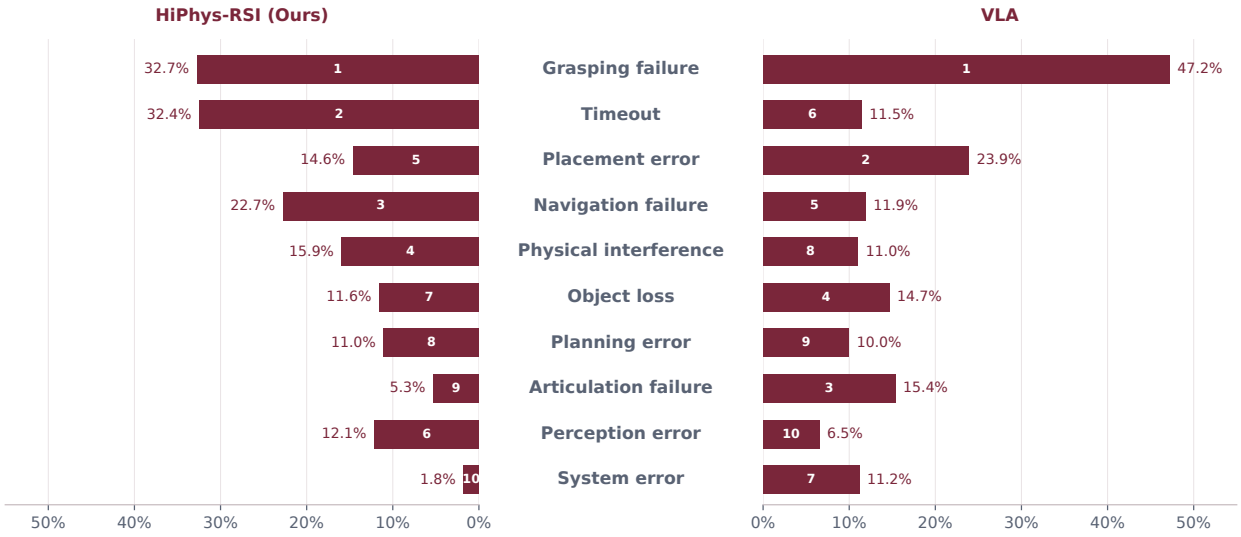


Figure 6 Comparison of failure modes between our approach and the VLA baseline. Bar lengths indicate the percentage associated with each failure mode, and numbers inside bars denote its frequency rank within each method. Detailed descriptions of each failure mode can be found at Section B. A task-specific failure mode heatmap is presented at Section C.

clearance. Under the current strategies, these constraints can lead to repeated unsuccessful attempts that consume the remaining execution budget.

6.5 Case Study: Unconventional Strategies in Agentic Pipelines

Beyond aggregate performance, we examine how pipeline construction and refinement produce manipulation strategies adapted to scene constraints and execution difficulties. Figure 7 presents six examples involving alternative contact points, preparatory actions, execution order, and placement geometry. Developed through agent proposals and human-guided refinement, these strategies illustrate how accumulated execution knowledge is translated into concrete robot behaviors.

Adapting grasps to object accessibility. In *cook_broccolini* (Task 96), the robot grasps the near rim of a frying pan when its handle points away or sideways. It lifts the pan level and carries it to the stove, avoiding a detour around the kitchen island to reach the handle. Similarly, in *carrying_out_garden_furniture* (Task 74), walls or nearby furniture can obstruct access to the wheelbarrow handles. The robot approaches from the wheel side, grasps the front tub rim with both hands, verifies the lift, and transports the wheelbarrow toward the doorway. Both strategies adapt contact selection to the object's accessibility within the scene.

Changing object configuration to facilitate manipulation. In *spray for bugs* (Task 38), directly grasping the upright sprayer is prone to collisions and can leave it held at an awkward angle for switching it on. The robot first tips the sprayer onto its side, making it easier to grasp from a suitable direction and orient for activation. In *stacking_wood* (Task 58), parallel placement can cause a log to roll or slide off its supporting log. The robot instead turns the carried log perpendicular to the supporting log and releases it slightly toward itself. One end rests on the supporting log while the other rests on the ground, allowing gravity and ground support to maintain contact and reduce the risk of rolling off. These behaviors modify object pose and support geometry to make subsequent interaction or placement more reliable.

Reordering actions and adapting transport modes. In *packing_meal_for_delivery* (Task 81), the robot first places three empty paper bags upright inside the box and then drops burgers into them. This sequence removes the need to hold each bag while inserting a burger, using the box to support the bags during filling. In *installing_a_scanner* (Task 92), the robot switches on the scanner while it rests on the desk, checks that the button turns green, and then slides it toward the laptop using a tilted closed hand coordinated with sideways base motion. When the scanner is too distant or poorly positioned for sliding, the pipeline uses a pick-and-carry procedure. These strategies simplify execution by changing when an operation is performed

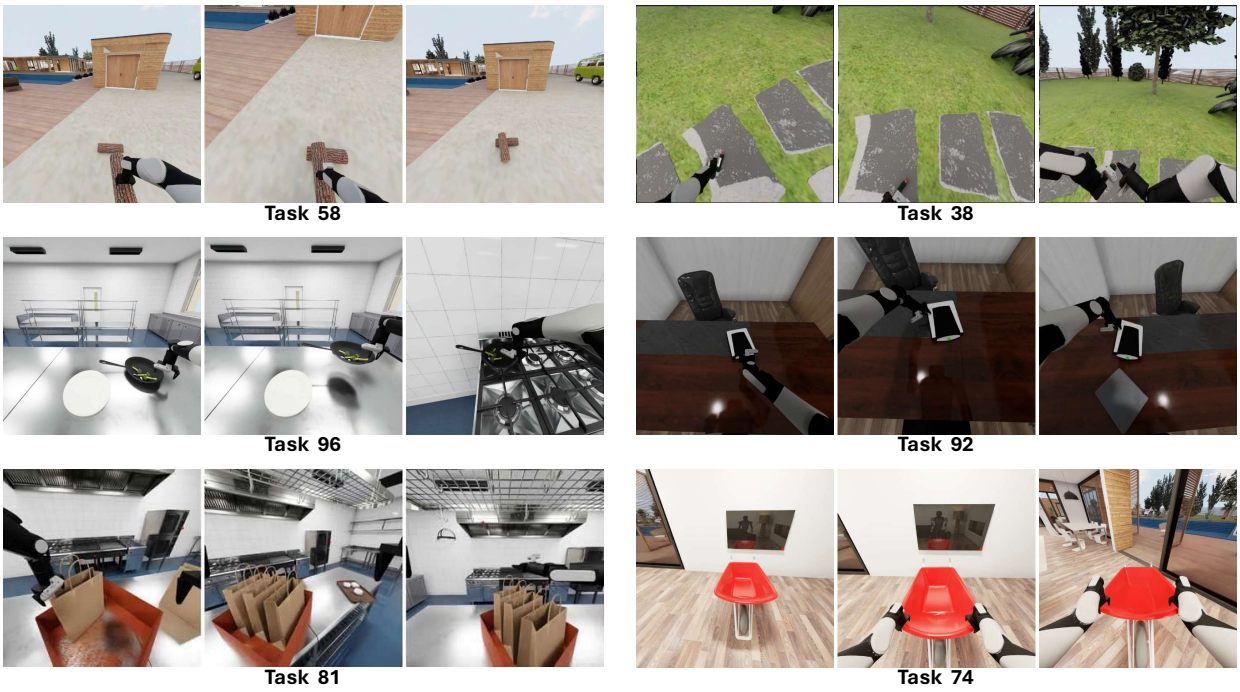


Figure 7 Unconventional Strategies in Agentic Pipelines. Representative executions illustrate unconventional manipulation strategies discovered during pipeline construction and refinement.

and how an object is moved.

Together, these cases illustrate how pipeline refinement captures practical adaptations in grasp selection, object configuration, and action coordination. The resulting procedures use scene geometry and physical support to address execution difficulties, with these adaptations stored in code for autonomous task execution.

6.6 Case Study: Benchmark Hacking or Reward-shaping?

Physical agentic recursive self-improvement raises concerns about benchmark hacking, but the boundary between exploiting an evaluation loophole and responding to reward shaping is not always clear. We use the official evaluator with matched task-execution budgets across methods, while hidden test instances remain inaccessible throughout development. These controls support consistent comparisons and evaluation of generalization beyond development instances.

Two cases in Figure 8 illustrate the blurred boundary between benchmark hacking and optimizing for the specified reward. In Task 58 (`stacking_wood`), the agent forms three separate pairs of logs rather than gathering them into a single woodpile. This satisfies the evaluator's requirement that each log touch another while circumventing the intended stacking objective, making it a relatively clear example of benchmark hacking. Task 48 (`canning_food`) is more ambiguous: the agent dices food directly inside the refrigerator, earning partial credit while leaving the subsequent transfer-to-bowl goals unmet. The evaluator correctly records incomplete progress, and cutting in place could constitute a valid intermediate step. However, the same behavior could also reflect optimization toward readily obtainable credit without progressing to full completion. Together, these cases show why benchmark hacking and reward optimization are difficult to separate: both respond to the specified objective, but their alignment with task intent ranges from clear circumvention to plausible yet incomplete progress.

This ambiguity is not unique to physical RSI; it reflects a longstanding reward-design problem discussed in the reinforcement learning community, where policies can optimize misspecified rewards without fulfilling the intended objective. As discussed by Amodei et al. (2016), a proxy that correlates with the intended objective under ordinary behavior may lose this correspondence under strong optimization. Similarly, success predicates and partial rewards that appear adequate for demonstration-like executions may become insufficient as RSI



Figure 8 Potential benchmark hacking behaviors in agent-constructed pipelines. Representative executions illustrate strategies that exploit underspecified success conditions or prioritize partial credit over task completion.

explores a broader space of strategies. These observations suggest that benchmark evaluation criteria could benefit from evolving alongside the systems optimized against them. Discovered shortcuts can inform this process by revealing where terminal conditions fail to capture the intended outcome or partial rewards favor strategies that stall before completion. When refining these criteria, it is also important to preserve legitimate alternative solutions, since deviation from demonstrations alone does not establish hacking. Improving task-solving capabilities and refining evaluation criteria can thus be complementary efforts: stronger systems expose specification gaps, and better criteria guide subsequent improvement toward the intended task. Complementing these efforts, better reconstruction and modeling of real-world environments and interactions in simulation could help align policy optimization and evaluation with deployment conditions, as explored in World Labs' work on real-to-sim-to-real pipelines (World Labs, 2026).

Beyond refining what counts as success, another useful direction is to evaluate the policy's ability to independently determine when success has been achieved. Even a well-specified success criterion may leave a capability gap unmeasured if the evaluator also tells the system when that criterion has been met. Future benchmarks could therefore consider stage-transition detection and completion checking as additional evaluation dimensions, for example by reserving simulator-state judgments for scoring while allowing the system to determine its own progress and termination. Assessing both accuracy and latency would be valuable, as repeated queries to large closed-source models can delay execution even when their assessments are correct. Dedicated work by our team explores this direction by distilling execution-state assessment into a small model that runs at over 1.7 Hz, enabling frequent checks of task progress and stage transitions (Huang et al., 2026a). Including such assessments alongside task outcomes could provide a fuller picture of whether a system can both achieve the intended goal and independently determine when to advance or stop.

7 Conclusion

In this report, we investigated scaling through structured, executable knowledge for robotic tasks, using a frozen VLM-based coding agent to construct and refine autonomous pipelines through HiPhys-RSI. Our hybrid system combines pipelines built from classical perception, planning, and control modules with inference-time routing between established pipelines and VLA policies. Systematic comparisons with three VLA variants spanning training duration and task-specific adaptation show broad performance advantages of our approach across the evaluated BEHAVIOR-1K task categories. These findings support structured knowledge accumulation as a complementary direction for scaling embodied intelligence, with coding agents automating part of the engineering required to compose and adapt robotic capabilities.

Our analysis suggests that structured scaling progresses from rapid gains through capability composition toward more demanding exploration of alternative strategies. Broad task coverage emerges relatively early, while difficult cases require disproportionate refinement effort. The failure mode analysis indicates that agentic systems can address many coordination and manipulation difficulties, but refinement alone remains limited by visibility, physical accessibility, and execution budgets. The case studies further illustrate how HiPhys-RSI captures unconventional strategies through changes in contact selection, object configuration, and action order. Future work should bridge the sim-to-real gap through a real–sim–real loop that uses physical observations and execution failures to calibrate simulations, refine pipelines and policies under realistic variations, and validate improvements on real robots. Deployment discrepancies could then guide further simulation updates, code revisions, and policy adaptation, extending HiPhys-RSI to real-world improvement.

References

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Paul J. Besl and Neil D. McKay. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992. doi: 10.1109/34.121791.

Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, 32(6):1309–1332, 2016.

Edward A Feigenbaum. On the thresholds of knowledge. *Foundations of artificial intelligence*, 47:185–250, 1992.

Edward A Feigenbaum et al. The art of artificial intelligence: Themes and case studies of knowledge engineering. 1977.

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.

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. *Proc. Int. Conf. Autom. Plan. Sched.*, 2020.

Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4 (1):265–293, 2021.

Elmer G. Gilbert, Daniel W. Johnson, and S. Sathiya Keerthi. A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988. doi: 10.1109/56.2083.

Jinbang Huang, Yuanzhao Hu, Zhiyuan Li, Ran Qi, Yixin Xiao, Yangzheng Wu, Tengyue Ba, Zhanguang Zhang, and Yingxue Zhang. Stageguard: Learning stage transitions for long-horizon robot tasks via agentic distillation. *arXiv preprint arXiv:2609.20791*, 2026a.

Jinbang Huang, Yuanzhao Hu, Zhiyuan Li, Ran Qi, Yixin Xiao, Zhanguang Zhang, Mark Coates, Tongtong Cao, and Yingxue Zhang. Roboharness: Memory-driven orchestration of heterogeneous robot policies for long-horizon planning. *arXiv preprint arXiv:2607.18060*, 2026b.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proc. Conf. on Robot Learning*, 2022.

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

Pablo Jiménez, Federico Thomas, and Carme Torras. 3d collision detection: a survey. *Computers & Graphics*, 25(2): 269–285, 2001.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE international conference on robotics and automation*. IEEE, 2011.

Oussama Khatib. A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, 1987. doi: 10.1109/JRA.1987.1087068.

John E Laird, Paul S Rosenbloom, and Allen Newell. Chunking in soar: The anatomy of a general learning mechanism. *Machine learning*, 1(1):11–46, 1986.

Ilia Larchenko, Gleb Zarin, and Akash Karnatak. Task adaptation of vision-language-action model: 1st place solution for the 2025 behavior challenge. *arXiv preprint arXiv:2512.06951*, 2025.

Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

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.

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*, 2026a.

Zexi Li, Yehang Zhang, Wenqian Li, Haojian Huang, Chenxu Wang, Shiyuan Deng, Yangkai Wei, Tianyi Zhang, Binghui Xie, Bohan Zhou, et al. Robo-harness k1: Harnessing robot-use agents via perception augmentation. *arXiv preprint arXiv:2609.29389*, 2026b.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as Policies: Language Model Programs for Embodied Control. In *IEEE International Conference on Robotics and Automation*, 2023. doi: 10.1109/ICRA48891.2023.10160591. URL https://arxiv.org/abs/2209.07753.

Jingsong Liang, Shuhao Liao, Shizhe Zhang, Diyuan Hou, Yuxin Cai, Xinjian Deng, Chengyang He, Wenhui Huang, Runjia Tan, Zhidong Wang, et al. Robofoundry: System-as-policy evolution for self-learning embodied agents. *arXiv preprint arXiv:2609.32862*, 2026.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Dingsheng Liu, Yangzheng Wu, Mahboubeh Asadi, Zhiyuan Li, Jinbang Huang, Yixin Xiao, Tongtong Cao, and Yingxue Zhang. Spatial grafting: Grounding 3d features for flow-matching robot policies. *arXiv preprint arXiv:2609.35249*, 2026.

David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60:91–110, 2004. doi: 10.1023/B:VISI.0000029664.99615.94.

Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, et al. Aspire: Agentic/skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026.

Ruben Mascaro and Margarita Chli. Scene representations for robotic spatial perception. *Annual Review of Control, Robotics, and Autonomous Systems*, 8(1):351–377, 2025.

John McCarthy et al. *Programs with common sense*. RLE and MIT computation center, Cambridge, MA, USA, 1960.

Allen Newell and Herbert A Simon. Computer science as empirical inquiry: Symbols and search. In *ACM Turing award lectures*, page 1975. 2007.

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2024.

Ran Qi, Yixin Xiao, Dawei Wang, Zhiyuan Li, Jinbang Huang, Xuan Zhao, Yintao Ma, Yangzheng Wu, Tongtong Cao, and Yingxue Zhang. RoboCoEvo: Expanding robot capabilities through agent–policy co-evolution.

Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. CHOMP: Gradient Optimization Techniques for Efficient Motion Planning. In *IEEE International Conference on Robotics and Automation*, 2009. doi: 10.1109/ROBOT.2009.5152817.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion Planning with Sequential Convex Optimization and Convex Collision Checking. *The International Journal of Robotics Research*, 2014. doi: 10.1177/0278364914528132.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Vítor Tinoco, Manuel F Silva, Filipe N Santos, Raul Morais, Sandro A Magalhães, and Paulo Moura Oliveira. A review of advanced controller methodologies for robotic manipulators: V. tinoco et al. *International Journal of Dynamics and Control*, 13(1):36, 2025.

Shivam Vats and Sayan Mondal. Logic geometric programming: An optimization-based approach to combined task and motion planning. 2020.

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.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*, 2024.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, volume 2025, pages 65882–65919, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Zimo Wen, Yijin Chen, Yuxuan Cao, Wendi Chen, Yanwen Zou, Wenye Yu, Fuhang Kuang, Han Xue, Jun Lv, Chuan Wen, et al. Roborsi: Stable, efficient, and reusable robot self-evolution in complex real-world environments. *arXiv preprint arXiv:2610.12424*, 2026.

World Labs. Building worlds that train robots, July 2026. URL https://www.worldlabs.ai/blog/real-to-sim-to-real.

Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, et al. Enpire: Agentic robot policy self-improvement in the real world. *arXiv preprint arXiv:2606.19980*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Xilun Zhang, Maggie Wang, Erik Bauer, Hong-Xing Yu, Huang Huang, Jiajun Wu, and Marco Pavone. Co-evolving robot orchestrators and policies through deployment. *arXiv preprint arXiv:2610.09228*, 2026a.

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

Zhigen Zhao, Shuo Cheng, Yan Ding, Ziyi Zhou, Shiqi Zhang, Danfei Xu, and Ye Zhao. A Survey of Optimization-Based Task and Motion Planning: From Classical to Learning Approaches. *arXiv preprint arXiv:2404.02817*, 2024. URL https://arxiv.org/abs/2404.02817.

Appendix

A Task Category Definitions.

We group tasks into eight categories according to their principal capability requirements.

(A) Mobile retrieval involves searching for, locating, and retrieving objects, including transporting them across rooms, doorways, or narrow passages.

(B) General placement involves moving objects onto surfaces or into broad containers without requiring a specific match or precise alignment.

(C) Matched placement requires selecting the appropriate destination for each object according to its identity, type, or task-defined correspondence, such as sorting bottles, cans, and paper into their respective bins.

(D) Precision manipulation requires careful control of object position and orientation for insertion, stacking, hanging, or attachment, such as mounting a camera on a tripod.

(E) Tool use involves manipulating a held tool to act on a target, including brushing, sweeping, spraying, vacuuming, or igniting.

(F) Material dispensing involves tilting a container to transfer its existing contents into another container or onto a target, such as pouring ingredients into a pan.

(G) Device operation involves operating mechanisms, buttons, or appliances to change a device or object's state, such as activating a washer or using a burner to cook food.

(H) Object transformation involves producing and tracking processed or split object forms, such as dicing an onion or splitting a log into halves.

B Failure Mode Definitions

We categorize execution failures into ten modes.

Grasping failure refers to an inability to acquire a stable grasp during pickup, including unsuccessful gripper closure or unreachable grasp configurations.

Placement error occurs when an object is placed or inserted at an incorrect location, orientation, or depth, failing to satisfy the required spatial relation.

Object loss denotes an unintended drop or loss after an object has been acquired, such as during transport or manipulation.

Articulation failure refers to unsuccessful interaction with an articulated object, such as failing to open or close a door, drawer, or lid to the required configuration.

Navigation failure occurs when the robot cannot locate or reach a target object or work area, including unsuccessful search, blocked routes, or inaccessible manipulation positions.

Timeout indicates that execution exhausts the available time budget before task completion, potentially due to prolonged retries, inefficient movement, or stalled execution.

Physical interference captures collisions, entrapment, unstable posture, or falls involving the robot or surrounding objects that impede execution.

Planning error refers to an inappropriate action sequence or decision that prevents progress, such as omitting prerequisites, executing actions in the wrong order, or persisting with an ineffective plan.

System error denotes disruptions attributable to the supporting software or evaluation environment, including runtime errors, simulator crashes, API failures, or simulation artifacts, when supported by the available evidence.

Perception error refers to incorrect interpretation of object identity, location, geometry, or state that leads to inappropriate target selection or actions.

These categories are not mutually exclusive: a rollout may exhibit multiple failure modes, and timeout may co-occur with the failure that caused the delay.

C Task-specific failure mode heatmap

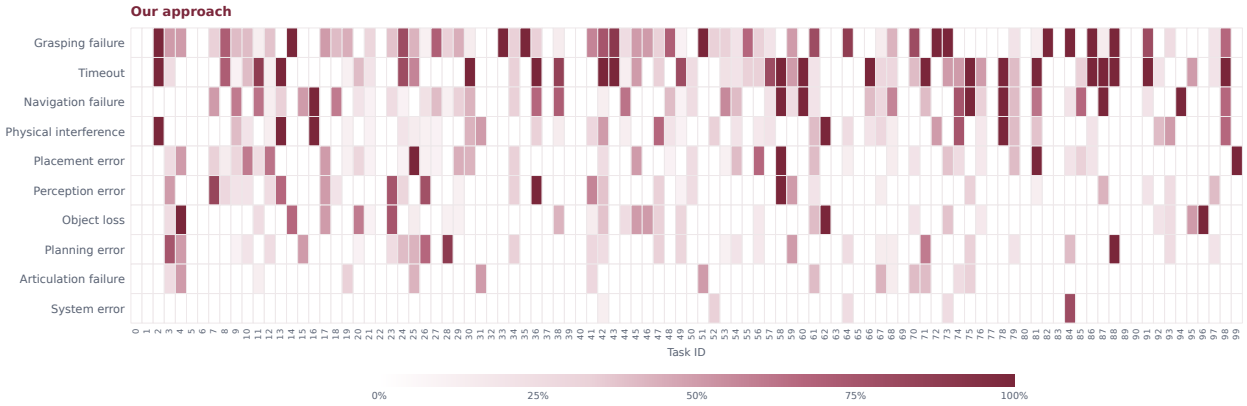


Figure 9 Per-task failure modes Columns correspond to task IDs and rows to failure categories. Color intensity indicates the frequency of each failure mode for each task, with darker cells representing higher percentages.