ROBOFOUNDRY: SYSTEM-AS-POLICY EVOLUTION FOR SELF-LEARNING EMBODIED AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

A foundation model should not act in isolation as an embodied agent. Yet, existing methods often optimize individual components of the agent stack, such as memory, context, skills, or action interfaces, rather than treating the supporting system itself as a unified policy. Moreover, interaction alone does not yield self-improvement unless execution experience is converted into persistent, validated system changes. We therefore propose **RoboFoundry**, the first embodied agentic framework that formulates this process as **Self-Evolving System-as-Policy**. RoboFoundry diagnoses capability gaps in *decision-making* and *memory management*, converts execution traces into validated task-specific system updates, and promotes recurring improvements to the general system. Evolution operates over two complementary surfaces: a context system that manages active internal context and persistent file-system memory, and a hierarchical skill system that organizes atomic skills, reusable compositions, and failure-conditioned recovery. A shared semantic interface separates embodiment-invariant decisions from embodiment-specific execution, allowing evolved system capabilities to transfer across heterogeneous robots. On EmbodiedBench, RoboFoundry achieves state-of-the-art performance, notably improving GPT-5.5 by $27.8\,\%$. It also brings Qwen3.7-Plus to near parity with GPT-5.5 ($70.3\,\%$ vs. $72.7\,\%$), showing consistent gains from system-as-policy evolution across foundation models. For long-horizon memory, RoboFoundry outperforms all baselines on RoboMemArena by at least $39.0\,\%$, even against methods assisted by external foundation models. On LIBERO-PRO, it further outperforms CapAgent0 by 243.8–$679.7\,\%$ across all perturbation types. In real-world deployments, RoboFoundry demonstrates *zero-shot transfer* and *online evolution* across robots and tasks, highlighting its potential for fully autonomous embodied agents.

1 INTRODUCTION

Recent advances in large language models (LLMs) and multimodal large language models (MLLMs) are shifting embodied intelligence from learning a separate policy per task toward using general-purpose foundation models as embodied decision makers. Early work mainly uses them as high-level planners, exposing observations, robot states, and admissible actions through designed prompts (Brohan et al., 2023; Huang et al., 2023; Jiang et al., 2023). As model outputs become increasingly executable, *code-as-policy* methods take the next step, composing perception and control primitives into programs that interact with the environment (Fu et al., 2026; Liu et al., 2026; Li et al., 2026; Zhang et al., 2026b), moving the foundation model from a planner inside the robotic stack toward the decision core of a closed-loop embodied agent.

Yet a foundation model alone does not define such an agent. The same model behaves very differently as the surrounding interaction structure changes: iterative observation, feedback, and verification support better-grounded decisions, while structured action interfaces reduce the burden of turning semantic decisions into executable behavior (Li et al., 2024; Liu et al., 2026; Kwok et al., 2026). *The executable embodied policy is therefore a system, rather than a model in isolation*, jointly determined by the model and by what information reaches it, what experience persists, and how its decisions become physical actions. Existing embodied agents, however, are rarely optimized as such a system, and instead refine a predefined component while leaving the rest unchanged: interaction harnesses improve model–environment feedback (Liu et al., 2026); memory systems improve access to past information (Huang et al., 2026; Zhang et al., 2026c); skill-learning agents

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

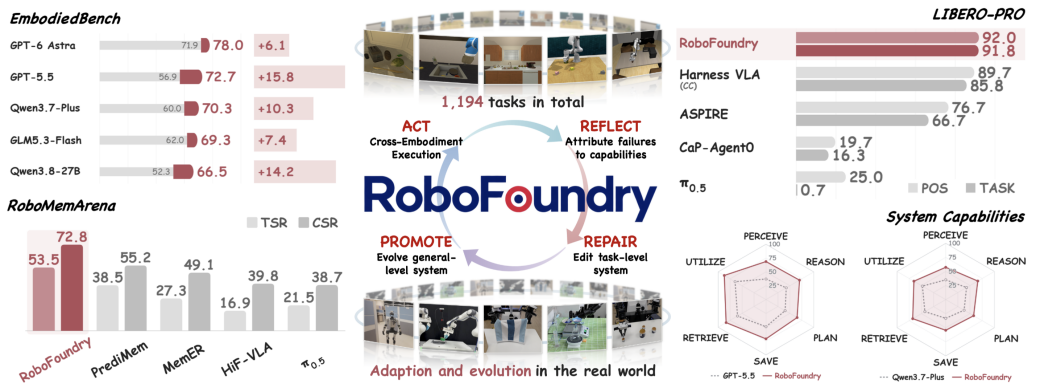

Figure 1: **RoboFoundry: System-as-Policy Evolution across benchmarks and real robots.** The central Act–Reflect–Repair–Promote loop executes across embodiments, attributes failures to capabilities, revises the task-level system, and promotes validated changes to the general system. The surrounding panels show gains across foundation models on EmbodiedBench, long-horizon memory results on RoboMemArena, and robustness perturbations under LIBERO-PRO. The capability plots compare GPT-5.5 and Qwen3.7-Plus before and after RoboFoundry, while the lower image strip illustrates adaptation and evolution on physical robots.

accumulate reusable behaviors (Wang et al., 2023; Lu et al., 2026; Wang et al., 2026); and coding agents refine programs from physical feedback (Xiao et al., 2026). Such gains can be substantial, but remain *component-local*: improving one component neither identifies another bottleneck nor determines how the improvement propagates through the system. **Richer interaction also does not by itself make an embodied system self-improving.** Most systems stay fixed after deployment even though interaction continuously exposes where their support is insufficient, so the same weakness recurs across episodes (Huang et al., 2023; Madaan et al., 2023; Brohan et al., 2023; Zhang et al., 2026c); The resulting experience is typically used for evaluation or manual debugging rather than converted into persistent, validated system changes (Li et al., 2024; Yang et al., 2025; Fu et al., 2026; Liu et al., 2026).

These observations lead to the central question: **Can an embodied agent treat its entire system as the policy, diagnose where that system limits behavior, and evolve the corresponding components from embodied experience?** To this end, we propose **RoboFoundry**, the first embodied agent framework that formulates the agent stack itself as a **Self-Evolving System-as-Policy**. Rather than committing in advance to one module to optimize, RoboFoundry lets embodied execution history decide which part of the system supporting the foundation model should change and how far that change should propagate, extending the progression from *language-model planning* to *code-as-policy execution*, and further to *system-as-policy evolution*.

Evolution operates over two complementary *surfaces*. The *context surface* governs how observations and accumulated experience are managed, spanning active internal context within an episode and persistent file-system memory across episodes. The *skill surface* organizes executable behavior, from atomic skills to reusable compositions and failure-conditioned recovery graphs. Execution traces decide both *why* the system is limited and *where* it should change: RoboFoundry attributes recurring failures to model-conditioned capability gaps, confines each revision to the responsible surface, and assigns it an explicit *scope*, where a task-specific repair is validated in place while a general update is promoted only after the same improvement proves effective beyond the held-in task. Self-evolution is therefore a controlled loop of diagnosis, intervention, validation, and promotion, rather than unconstrained self-rewriting.

A shared semantic interface keeps evolved capability from being bound to one robot or one backbone: decisions are expressed once as semantic units, while replaceable embodiment bindings realize them through robot-specific primitives. Because evolution acts on the semantic side, a system evolved on one platform is reused on another by swapping its bindings, and the same evolved support attaches to different foundation models.

Multiple embodied benchmarks test whether system-level evolution improves decision-making, long-horizon memory, and robustness under task and environment perturbations, repeated across foundation models of different capability to verify that the gains come from the system rather than from a particular backbone; real-robot deployments then test transfer to unseen robots and tasks and continued improvement after deployment. RoboFoundry achieves state-of-the-art results in all settings and keeps evolving across simulation and the real world.

In summary, we make key contributions as follows:

- We propose **RoboFoundry**, the first embodied agent framework that formulates the agent system itself as the policy and makes it *self-evolving*, advancing embodied agents from *code-as-policy execution* to **system-as-policy evolution**.
- We develop a capability-guided mechanism for context–skill co-evolution that turns execution traces into targeted, validated system revisions and promotes recurring improvements from task-specific support to the general system.
- We introduce a shared semantic interface that decouples embodiment-invariant decisions from embodiment-specific execution, making evolved capability reusable across heterogeneous robots and tasks.
- RoboFoundry delivers **substantial gains across multiple benchmarks**, spanning embodied decision-making, long-horizon memory, and robustness to perturbations, brings open-source backbones close to frontier-model performance, and demonstrates **zero-shot generalization and online evolution** in real-world robotic experiments.

2 METHOD

2.1 SELF-EVOLVING SYSTEM AS POLICY

RoboFoundry treats the overall system supporting task execution as the policy to optimize. Let M denote a frozen foundation model and $H(M)$ its supporting system. Rather than updating the parameters of M, RoboFoundry improves the effective capability of the model–system pair by evolving $H(M)$ through filesystem operations. The model inspects the supporting system using operations such as `cat` and `grep`, and edits it using `add`, `modify`, and `delete`. We decompose the supporting system into a task-specific component H_t and a general component H_g, with corresponding optimization spaces S_t and S_g. The task-specific space S_t comprises semantic-level task formulation S_t^c and embodiment-specific execution S_t^e.

RoboFoundry organizes system evolution as an inner–outer loop. The **inner loop** executes tasks under the current system and collects execution traces. For a task instance $x \sim \mathcal{X}$, a rollout is written as $\tau \sim \mathcal{T}(H_g, H_t, x)$, where τ records system and tool calls, step-wise observations and robot states, execution outcomes, and task feedback. Here, $\mathcal{T}(H_g, H_t, x)$ denotes the set of stored traces. The **outer loop** proposes system edits, which are deployed and evaluated through subsequent inner-loop rollouts. We detail task execution in Section 2.2.

The outer loop first performs *task-level system improvement* over S_t to improve completion of the current task. Here, $r_x(\Delta H_t, \tau)$ denotes the performance score of rollout τ on task x, where ΔH_t is a candidate edit to the task-level system. Based on execution traces, the outer loop revises either the semantic-level system in S_t^c or the embodiment-specific execution system in S_t^e:

$$H_t^* = \arg\max_{H_t \in S_t} \mathbb{E}_\tau \left[r_x(\Delta H_t, \tau) \right]. \tag{1}$$

Beyond the current task, RoboFoundry performs *general-level system improvement* over the persistent system H_g. It examines whether a successful task-level improvement addresses a capability gap shared by other tasks, using traces accumulated across tasks and embodiments. Let $\mathcal{X}_\mathcal{A}$ denote the task distribution associated with these capability gaps and ΔH_g denotes the general-level edit. General-level optimization is formulated as

$$H_g^* = \arg\max_{H_g \in S_g} \mathbb{E}_{x \sim \mathcal{X}_\mathcal{A},\, \tau \sim \mathcal{T}(H_g, H_t^*(x), x)} \left[r_x(\Delta H_g, \tau) \right]. \tag{2}$$

The outer loop promotes improvements supported by the broader trace history into H_g, where they persist across subsequent tasks. The objectives above specify what each level seeks to improve; the

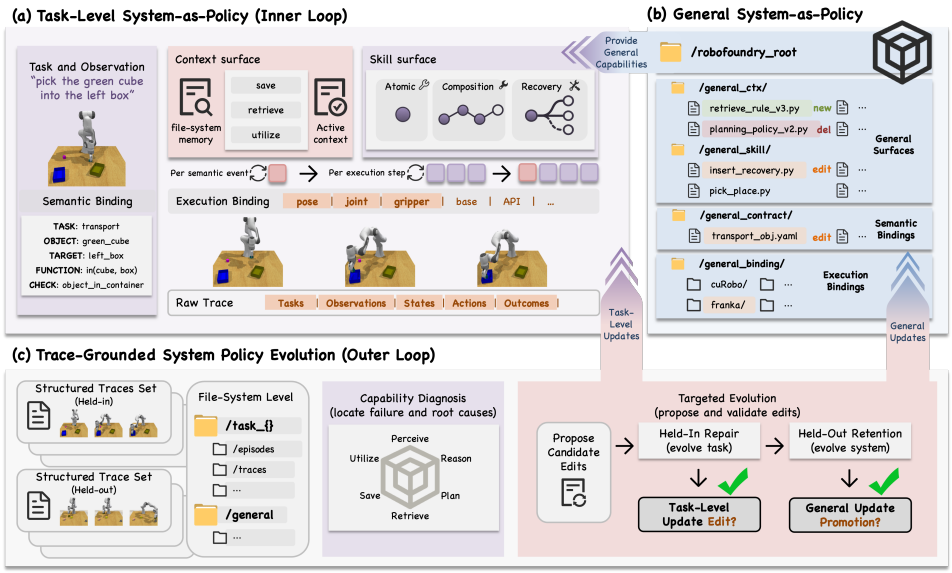

Figure 2: **Overview of RoboFoundry.** (a) The inner loop runs task-specific System-as-Policy through context/skill surfaces and semantic/execution bindings, producing embodiment-specific rollouts and traces. (b) The general system supplies reusable context and skill rules to task-level systems. (c) The outer loop diagnoses capability failures and validates targeted edits through held-in repair and held-out retention. Accepted edits update the task-specific system or promote the general system.

trace-grounded edits below provide the local improvement procedure. Thus, although M remains frozen, the effective capability of the model–system pair can evolve through persistent changes to its supporting system. Algorithm 1 summarizes the overall execution–evolution loop. We detail diagnosis and improvement in Section 2.3.

2.2 Inner-Loop System Execution

RoboFoundry uses the same task-execution loop in simulation and the real world. Given a task description, observations, and robot states, cross-embodiment execution requires separating embodiment-invariant task semantics from embodiment-specific kinematic and dynamic primitives. We introduce a semantic binding layer $H_t^s(M) = \texttt{BIND}_s(H_t(M))$, which exposes task logic shared across embodiments. Accordingly, $H_t^s(M)$ represents, reasons about, and decomposes the task without directly handling robot-specific constraints. An execution binding layer $H_t^e(M) = \texttt{BIND}_e(H_t(M))$, grounds the resulting semantic decisions in embodiment-specific execution. This separation helps distinguish errors in task reasoning from failures in execution. The execution backend can use frozen VLA policies, functions generated by coding agents, visuomotor APIs such as CuRobo, or other foundation-model-supported robotic interfaces.

RoboFoundry exposes two task-level surfaces, *context* and *skill*, for semantic-level optimization in $H_t^s(M)$. (1) Context: The context system consists of persistent filesystem memory and the active context presented to the model. Rather than updating memory at every step, $H_t^s(M)$ monitors task progress and updates context when an event warrants it. We define the context operation space as $S_t^{\text{ctx}} = \{\texttt{SAVE}, \texttt{RETRIEVE}, \texttt{UTILIZE}\}$. Given execution histories of observations, actions, and feedback, `SAVE` selects task-relevant evidence for persistent storage. What should be retained depends on task semantics: an occlusion task may require object and spatial relations, whereas a counting task may require occurrences of relevant actions. `RETRIEVE` selects stored evidence for the next decision, and `UTILIZE` incorporates it into the active context. These operations are recorded in the execution trace as evidence for subsequent system improvement.

(2) Skill: $\texttt{BIND}_s$ separates task logic from embodiment-specific constraints. We organize the skill surface hierarchically into atomic skills, skill compositions, and recovery. Atomic skills

Algorithm 1 RoboFoundry: system evolution and task execution

Overall evolution

Require: General system H_g, task-level system H_t, tasks $\mathcal{X}$, trace set $\mathcal{T}$

$\mathcal{T} \leftarrow \emptyset$
for each task $x \in \mathcal{X}$ **do**
  for each rollout on x **do**
    $\tau \leftarrow \textsc{RunTask}(H_g, H_t, x)$
    $(H_g, H_t, \mathcal{T}) \leftarrow \textsc{Revise}(H_g, H_t, \mathcal{T}, \tau)$
return $H_g, H_t, \mathcal{T}$

RunTask(H_g, H_t, x)

$H_t^s \leftarrow \texttt{BIND}_s(H_t)$; $H_t^e \leftarrow \texttt{BIND}_e(H_t)$
Initialize state z_0, memory $\mathcal{M}_0$; $\tau \leftarrow \emptyset$
for each step t until completion **do**
  $c_t \leftarrow \textsc{Utilize}(\textsc{Retrieve}(x, z_t, \mathcal{M}_t), z_t)$
  $p_t \leftarrow M(x, z_t, c_t; H_g, H_t^s)$
  $(z_{t+1}, f_t) \leftarrow \textsc{Execute}(H_t^e(p_t))$
  $\tau \leftarrow \tau \,\|\, (z_t, c_t, p_t, z_{t+1}, f_t)$
  $\mathcal{M}_{t+1} \leftarrow \textsc{Save}(\mathcal{M}_t, \tau)$ **on event**
return τ

are indivisible executable units, such as `grasp` or `move-to`, whose low-level implementations are initially handcrafted and verified. For a given task, atomic skills are composed into reusable procedures that preserve its semantic logic; for example, `move-to`, `grasp`, and `place` can form a transfer procedure. During the inner loop, RoboFoundry maintains a recovery tree with key states as nodes and skill executions, including failed transitions, as edges. Upon failure, $H(M)$ queries the tree to identify a valid state from which to replan, rather than repeating the failed action. For example, if a cube falls during transfer, the system can use the state where the cube rests on the support surface to plan another grasp. Execution traces are therefore attributable to atomic execution, composition, or recovery, providing evidence for skill improvement.

2.3 Trace-Grounded Diagnosis and System Improvement

Task-level system improvement. Given an execution trace τ, the task-level system policy $H_t(M)$ diagnoses the observed failure or inefficiency by attributing it to one of six capabilities: $\mathcal{A} = \{\texttt{perceive}, \texttt{reason}, \texttt{plan}, \texttt{save}, \texttt{retrieve}, \texttt{utilize}\}$. The first three characterize decision-making, while the latter three characterize memory management. $H_t(M)$ then localizes the capability gap and proposes an edit ΔH_t to the corresponding context or skill surface. The candidate is evaluated in a new inner-loop execution. If it resolves the failure, we denote the successful edit by $\Delta H_t^* = \Delta H_t$ and commit it as $H_t^* \leftarrow H_t \oplus \Delta H_t^*$, where $\oplus$ denotes applying an edit to the current system. The resulting trace, including ΔH_t^* and its execution feedback, is stored in $\mathcal{T}$ and indexed by capability $a \in \mathcal{A}$ for later general system improvement.

General system improvement. Task-specific corrections cannot guarantee performance on similar cases when the underlying general capability remains unchanged. RoboFoundry therefore evaluates whether a successful task-level edit can be generalized into a reusable system-level rule.

For each successful repair, its stored trace τ_i records the task x_i, observations and robot states before and after the repair, pre-repair history, committed task-level edit $\Delta H_{t,i}^*$, and execution feedback. We select successful records of the same capability from other tasks as the held-out trace set $\mathcal{T}_a^{\text{out}}(x) = \{\tau_i \in \mathcal{T} : a_i = a,\ x_i \neq x\}$.

From the successful edit ΔH_t^* on task x and its trace τ_x, we abstract a candidate general edit $\Delta H_g \leftarrow \text{Abstract}(\Delta H_t^*, \tau_x; H_g)$. For each held-out trace, let τ_i^{pre} denote its recorded task, observation, state, and pre-repair history. We replay this information and use the candidate general system to propose a task-level edit $\Delta H_{t,i}^{'} \leftarrow \text{Propose}(H_g \oplus \Delta H_g, \tau_i^{\text{pre}})$. The system policy $H(M)$ then acts as a verifier. Given the candidate $\Delta H_{t,i}^{'}$, the recorded successful edit $\Delta H_{t,i}^*$, and its execution feedback, it scores whether the candidate addresses the same capability gap. Let $r_x(\Delta H_t, \tau_i)$ denote this score. We measure capability-conditioned transfer as

$$\Delta_{\text{cap}}(\Delta H_g) = \frac{1}{|\mathcal{T}_a^{\text{out}}(x)|} \sum_{\tau_i \in \mathcal{T}_a^{\text{out}}(x)} \left[r_x(\Delta H_{t,i}^{'}, \tau_i) - r_x(\Delta H_{t,i}^*, \tau_i) \right]. \tag{3}$$

We promote a candidate ΔH_g^* when $\Delta_{\text{cap}}(\Delta H_g^*) \geq 0$ and commit it as $H_g^* \leftarrow H_g \oplus \Delta H_g^*$. Thus, task-level evaluation verifies whether an edit repairs the observed failure, while capability-conditioned held-out evaluation assesses whether the general edit addresses the same capability gap across other tasks.

3 EXPERIMENTS

3.1 SYSTEM-LEVEL EVOLUTION OF EMBODIED DECISION-MAKING

EmbodiedBench (Yang et al., 2025) evaluates embodied decision-making across 1,128 tasks in four environments: EB-ALFRED and EB-Habitat test high-level semantic planning, while EB-Navigation and EB-Manipulation require low-level executable actions. We instantiate the raw backbone, *RoboFoundry-Lite*, and full *RoboFoundry* across multiple frontier foundation models: GPT-6 Astra, GPT-5.5, Qwen3.7-Plus, Qwen3.8-27B, and GLM5.3-Flash. Lite ablates general evolution and keeps task edits only. All variants share the same task stream and interaction budget, evolve from the first scored episode, and retain every evolution trajectory in the reported score.

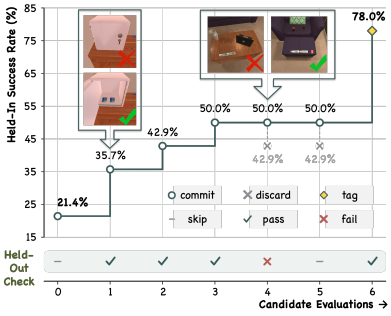

Figure 3: **Example of evolution trace on the EB-Habitat spatial subset.** Here, RoboFoundry (GPT-5.5) rises Held-in success rate from 21.4 % to 78.0 % over six candidate evaluations. The final version passes the held-out check and is promoted to the general system.

Table 1 shows consistent gains across all five backbones, indicating that RoboFoundry is compatible with heterogeneous foundation models and repairs backbone-specific bottlenecks rather than converging to a fixed harness. It improves GPT-5.5 by 27.8 %, and notably lifts Qwen3.7-Plus to near parity with RoboFoundry (GPT-5.5), showing that performance is governed by the evolved system rather than dominated by a stronger backbone. Even the frontier-tier GPT-6 Astra benefits from RoboFoundry, with an 8.5 % relative gain. The 11.2 % relative gain of full RoboFoundry over RoboFoundry-Lite on Qwen3.7-Plus further isolates general-scope evolution as the critical ingredient. Subset results are detailed in Tables 5–8. We further evaluate a 16K-token context budget in Table 9. A larger budget itself does not consistently improve task success (e.g., GLM5.3-Flash drops from 62.0% at 4K to 61.3% at 16K), while RoboFoundry still improves all backbones at 16K by 7.9 %–13.1 % relative to their baselines.

Figure 3 further traces one evolution run on the EB-Habitat spatial subset: three committed edits lift held-in success from 21.4 % to 50.0 %, two candidates are discarded without changing the system, and the last is tagged for promotion, reaching 78.0 %. We further investigate how RoboFoundry scales with the amount of held-in data used for evolution, reserving 50 held-out tasks per suite for testing. As shown in Figure 4, evolution is data-efficient: one quarter of the held-in tasks already reaches 78.0 % on EB-ALFRED and 66.0 % on EB-Habitat, and three quarters recover most of the final score, with EB-Habitat saturating at 74.0 %.

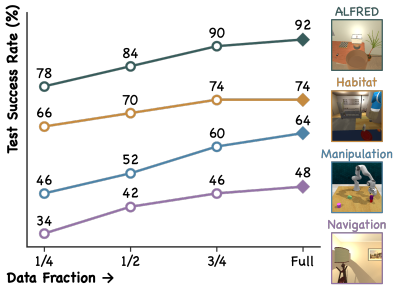

Figure 4: **Example of scaling with held-in data on EmbodiedBench.** RoboFoundry (GPT-5.5) is evaluated on 50 fixed held-out tasks per suite. Using three quarters of the held-in data already matches the full-data success rate of 74.0 % on EB-Habitat, highlighting data-efficient evolution.

3.2 LONG-HORIZON MEMORY THROUGH CONTEXT EVOLUTION

RoboMemArena (Lei et al., 2026) evaluates long-horizon memory across 26 tasks in four categories: **T**ransfer, **O**cclusion, **C**ounting, and **S**equence. We follow the official protocol and report task success rate (TSR) and cumulative success rate (CSR). RoboFoundry uses Qwen3.7-Plus as the backbone for the main comparison. Evolution begins with the first scored episode, and all trajectories used for updates count toward the evaluation budget. Task state is reset between trials; only validated task-agnostic context revisions persist. Table 2 shows that RoboFoundry achieves 53.5 % TSR and 72.8 % CSR, outperforming all baselines across all four categories. Compared with PrediMem, RoboFoundry achieves its largest relative TSR gain on Transfer, improving it by 213.3 %.

Table 1: **Overall comparison on EmbodiedBench.** Avg. is the arithmetic mean of success rates across the four suites. All results are reported in percentage (%). Gains are reported in percentage points (+pp).

Method	Avg.	EB-ALFRED	EB-Habitat	EB-Navigation	EB-Manipulation
RoboFoundry (GPT-6 Astra)	**78.0**$^{+6.1}$	**90.0**$^{+2.7}$	86.7$^{+11.4}$	**80.2**$^{+3.2}$	**54.9**$^{+6.9}$
RoboFoundry (GPT-5.5)	72.7$^{+15.8}$	84.0$^{+7.3}$	**88.7**$^{+24.7}$	72.0$^{+17.3}$	45.9$^{+13.8}$
RoboFoundry (Qwen3.7-Plus)	70.3$^{+10.4}$	81.3$^{+8.0}$	80.3$^{+15.3}$	71.9$^{+7.6}$	47.7$^{+10.5}$
RoboFoundry (GLM5.3-Flash)	69.3$^{+7.4}$	80.0$^{+8.3}$	75.0$^{+14.3}$	72.7$^{+1.7}$	49.5$^{+5.1}$
RoboFoundry (Qwen3.8-27B)	66.5$^{+14.2}$	78.0$^{+12.3}$	77.0$^{+19.0}$	68.3$^{+16.0}$	42.8$^{+9.5}$
RoboFoundry-Lite (GPT-5.5)	65.5$^{+8.6}$	78.0$^{+1.3}$	74.3$^{+10.3}$	69.7$^{+15.0}$	39.9$^{+7.8}$
RoboFoundry-Lite (Qwen3.7-Plus)	63.2$^{+3.2}$	75.0$^{+1.7}$	72.0$^{+7.0}$	68.0$^{+3.7}$	37.6$^{+0.4}$
GPT-6 Astra	71.9	87.3	75.3	77.0	48.0
GLM5.3-Flash	62.0	71.7	60.7	71.0	44.4
Qwen3.7-Plus	60.0	73.3	65.0	64.3	37.2
GPT-5.5	56.9	76.7	64.0	54.7	32.1
Qwen3.8-27B	52.3	65.7	58.0	52.3	33.3
Claude-3.5-Sonnet	50.5	64.0	68.0	44.7	25.4
GPT-4o	50.4	56.3	59.0	57.7	28.5
Claude-3.7-Sonnet	49.9	67.7	58.7	45.0	28.3

Table 2: **Long-horizon memory results on RoboMemArena.** RoboFoundry achieves the best TSR and CSR overall and across all four categories. Each entry reports TSR / CSR (%).

Method	Overall	Transfer	Occlusion	Counting	Sequence
RoboFoundry	**53.5 / 72.8**	**70.5 / 78.1**	**38.3 / 57.6**	**55.2 / 82.5**	**75.2 / 92.1**
PrediMem	38.5 / 55.2	22.5 / 45.2	27.3 / 38.4	45.7 / 69.3	72.5 / 89.5
MemER	27.3 / 49.1	20.0 / 36.1	16.4 / 33.2	27.1 / 65.1	65.0 / 79.1
HiF-VLA	16.9 / 39.8	17.5 / 38.9	12.7 / 27.1	8.6 / 45.9	42.5 / 70.2
$\pi_{0.5}$	21.5 / 38.7	20.0 / 42.8	12.7 / 17.2	14.3 / 50.9	60.0 / 71.6
MemoryVLA	15.0 / 35.3	15.0 / 37.2	7.3 / 13.1	14.3 / 55.1	37.5 / 65.2

We further apply RoboFoundry to standalone $\pi_{0.5}$ (Black et al., 2025) and PrediMem, which combines Qwen3-VL-8B-Instruct for explicit memory management and planning with $\pi_{0.5}$ for execution. Table 3 shows improvements in both TSR and CSR across all categories for both architectures, with a $200.0\,\%$ relative TSR gain for PrediMem on Transfer. These results support the applicability of RoboFoundry's context evolution across different memory architectures.

3.3 System Evolution under Distribution Shift

We use LIBERO-PRO (Zhou et al., 2025) to test whether persistent system evolution improves robustness to changes in object layouts and task specifications. We compare against CaP-Agent0 (Fu et al., 2026), ASPIRE (Lu et al., 2026), and both Harness VLA variants (CodeX and CC) (Zhang et al., 2026c). Following the CaP-Agent0 protocol, we evaluate 30 tasks from the Object, Goal, and Spatial suites under *Position* and *Task* perturbations, using the same perception and control primitives, multi-turn interaction setting, and 50 trials per task and perturbation. Evolution begins with the first scored rollout, and all rollouts used for updates count toward the evaluation budget.

Table 4 shows that RoboFoundry achieves the highest average success rate across the six settings among methods without privileged object poses ($91.9\,\%$ vs. $87.8\,\%$ for Harness VLA (CC)). It also outperforms CaP-Agent0 with up to a $679.7\,\%$ relative gain on Spatial position perturbations, supporting the benefit of persistent system evolution beyond within-episode program repair. We report task-wise results in Tables 10–12.

To assess how much perception limits the evolved system, we additionally provide RoboFoundry† with privileged simulator object poses. Success improves across all six settings, reaching **100%** on both Object and Goal task perturbations. These gains suggest that perception remains a bottleneck and that RoboFoundry can translate more accurate state information into more reliable task execution.

Table 3: **Context evolution across agent configurations.** RoboFoundry improves both standalone $\pi_{0.5}$ and PrediMem across all four categories. Each entry reports TSR/CSR (%).

Memory Category	$\pi_{0.5}$		PrediMem	
	Base	+RoboFoundry	Base	+RoboFoundry
Transferring	20.0 / 42.8	**62.5 / 71.2**	22.5 / 45.2	**67.5 / 76.2**
Occlusion	12.7 / 17.2	**33.6 / 47.2**	27.3 / 38.4	**36.4 / 56.4**
Counting	14.3 / 50.9	**58.8 / 79.6**	45.7 / 69.3	**53.7 / 80.3**
Sequence	60.0 / 71.6	**70.0 / 90.8**	72.5 / 89.5	**73.0 / 90.5**
Avg.	26.8 / 45.6	**56.2 / 72.2**	42.0 / 60.6	**57.7 / 75.9**

Table 4: **Results on LIBERO-PRO under perturbations.** Each entry reports position/task success rate (%). † denotes privileged simulator object poses.

Method	Object	Goal	Spatial
OpenVLA	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0
π_0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0
$\pi_{0.5}$	17.0 / 1.0	38.0 / 0.0	20.0 / 1.0
CaP-Agent0	21.8 / 18.2	25.6 / 16.8	11.8 / 14.0
ASPIRE	**98.0 / 95.0**	**81.0 / 45.0**	**51.0 / 60.0**
Harness VLA (Codex)	**81.0 / 69.0**	**94.0 / 91.0**	**75.0 / 66.0**
Harness VLA (CC)	**94.0 / 80.0**	**88.0 / 90.0**	**87.0 / 87.5**
RoboFoundry	**96.0 / 98.0**	**88.0 / 86.0**	**92.0 / 91.5**
RoboFoundry†	99.0 / 100.0	96.0 / 100.0	98.0 / 99.2

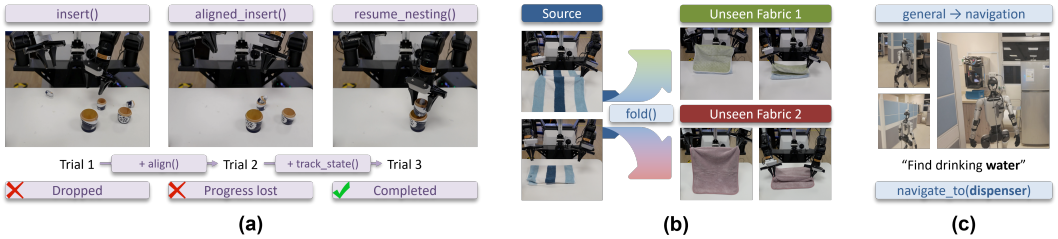

Figure 5: **Real-world evolution and transfer of RoboFoundry.** (a) Failures motivate alignment and progress-tracking edits, enabling nesting-doll completion in Trial 3. (b) Towel-folding skills evolved on the blue towel transfer to unseen green and pink fabrics. (c) The general system grounds a semantic navigation goal in executable actions on a Unitree G1 humanoid.

3.4 Real-World Evaluation

We evaluate whether the general RoboFoundry system transfers to physical robots and continues improving after deployment. Throughout the real-world experiments, RoboFoundry takes GPT-5.5 as the supporting foundation model and $\pi_{0.5}$ as its manipulation backend, following (Lei et al., 2026). For a fair comparison, RoboFoundry and the standalone baseline $\pi_{0.5}$ share the same fine-tuned checkpoint on corresponding tasks. RoboFoundry starts without task-specific system adaptation. We report the success rate of 10 trials per task, including all trials used for online evolution, with system updates applied only between trials.

We first evaluate online evolution on an AgileX COBOT MAGIC platform. As shown in Figure 5(a), nesting-doll manipulation requires both small-to-large ordering and precise insertion. Early failures drive complementary skill and context revisions: alignment checks improve insertion, while progress tracking enables execution to resume from the last completed subtask. Finally, RoboFoundry achieves 70 % success, compared with 30 % for standalone $\pi_{0.5}$. Moreover, Figure 5(b) demonstrates zero-shot transfer of the evolved folding skill: RoboFoundry evolves only on the blue towel and transfers directly to held-out green and pink towels with different appearance, geometry, and material properties. Without further adaptation, RoboFoundry achieves over 80 % success, compared with below 20 % for $\pi_{0.5}$, which needs further fine-tuning. Figure 5(c) extends RoboFoundry beyond manipulation to semantic navigation on a Unitree G1 humanoid. Through the shared semantic interface, RoboFoundry maps "Find drinking water" to a dispenser identified in RGB observations, while embodiment-specific bindings translate this target into navigation actions.

As shown in Figure 6, we also evaluate a long-horizon chemistry experiment with six dependent subtasks. RoboFoundry successfully preserves subtask state through its context surface and coordinates execution through reusable skill compositions and recovery structures. Crucially, completed steps become persistent prerequisites for later actions: the final pouring step retrieves the funnel-ready

state recorded several subtasks earlier. On the other hand, standalone $\pi_{0.5}$ struggles to complete the full procedure.

4 Related Work

Foundation Models as Embodied Agents. LLMs and MLLMs increasingly guide embodied decision-making. Early work connects language and multimodal observations to robot actions through vision-language-action models, multimodal prompts, and feedback-guided planning (Brohan et al., 2023; Jiang et al., 2023; Huang et al., 2023). Code-as-policy methods further enable models to compose perception and control primitives into executable programs (Fu et al., 2026; Zhang et al., 2026b). The same backbone can behave differently depending on its surrounding interaction harness (Liu et al., 2026; Zhang et al., 2026c), verification (Kwok et al., 2026), and embodied decision interface (Li et al., 2024). RoboFoundry formalizes this dependence as *System-as-Policy*: the executable policy is jointly determined by the foundation model and its supporting system. It makes this system an explicit target of evolution, turning embodied experience into persistent changes to future behavior.

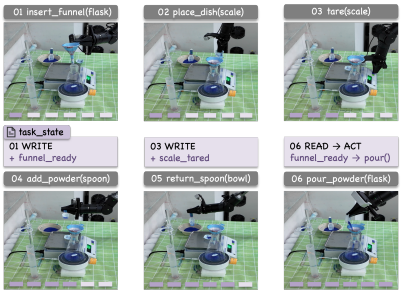

Figure 6: **Long-horizon chemistry manipulation with RoboFoundry.** A task-state record connects six dependent subtasks: funnel readiness and scale taring are written during execution, and the final pouring step reads the earlier funnel-ready state to guide skill execution.

Self-Evolving LLM/MLLM Agents. A parallel line improves agents through reflective self-correction (Madaan et al., 2023; Shinn et al., 2023), persistent memory (Packer et al., 2023; Zhao et al., 2024), and automated search over prompts, workflows, or code (Khattab et al., 2023; Hu et al., 2025; Zhang et al., 2026a). RoboFoundry extends system-level adaptation to cross-embodied execution. Its shared semantic interface separates task-level reasoning from robot-specific execution, allowing system revisions to be evaluated and reused across embodiments.

Self-Improving Embodied Agents. Existing work studies skill acquisition (Wang et al., 2023; Lu et al., 2026; Wang et al., 2026), memory-guided execution and policy orchestration (Huang et al., 2026; Zhang et al., 2026c; Lei et al., 2026), program repair with execution feedback (Fu et al., 2026), and closed-loop policy improvement (Li et al., 2026; Xiao et al., 2026). RoboFoundry treats context and skills as complementary intervention surfaces of the executable system policy. It evaluates task-level repairs on related held-out cases before promoting them into general system capabilities that support subsequent tasks.

5 Conclusion

We present RoboFoundry, an embodied agentic framework for *System-as-Policy Evolution*. Acting as a bridge between general-purpose models and embodied scenarios, RoboFoundry improves the embodied agent lifecycle, from decision-making and memory management to skill execution. It uses execution traces to improve task performance and promotes validated improvements into general system capabilities that benefit future tasks. RoboFoundry demonstrates strong autonomy and evolvability across multiple benchmarks and diverse real-world tasks. Looking forward, we aim to build an embodied flywheel that connects simulation and real-world experience, transfers knowledge from diverse non-embodied data to embodied tasks, and enables system evolution and model learning to reinforce each other.

AI USE STATEMENT

In this work, we used generative AI tools to improve the language and readability of the manuscript. We also accessed foundation models through APIs to conduct experiments that validate our proposed framework and support our contributions. The research ideas, framework, methodology, experimental design, and interpretation of results were developed independently by the authors. We reviewed and revised all AI-assisted text for accuracy and consistency with our work. We take responsibility for the final content, claims, and results of this work.

REFERENCES

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 27–30 Sep 2025.

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, pp. 287–318. Pmlr, 2023.

Max Fu, Justin Yu, Karim El-Refai, Ethan Kou, et al. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026. URL https://arxiv.org/abs/2603.22435.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

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*, 2026.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, et al. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 14–18 Dec 2023.

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, et al. VIMA: Robot manipulation with multimodal prompts. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 14975–15022. PMLR, 23–29 Jul 2023.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Jacky Kwok, Shulu Li, Pranav Atreya, Yuejiang Liu, Yixing Jiang, Chelsea Finn, Marco Pavone, Ion Stoica, and Azalia Mirhoseini. Llm-as-a-verifier: A general-purpose verification framework. *arXiv preprint arXiv:2607.05391*, 2026.

Huashuo Lei, Wenxuan Song, Huarui Zhang, Jieyuan Pei, Jiayi Chen, Haodong Yan, Han Zhao, Pengxiang Ding, Zhipeng Zhang, Lida Huang, et al. Robomemarena: A comprehensive and challenging robotic memory benchmark. *arXiv preprint arXiv:2605.10921*, 2026.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li E Li, Ruohan Zhang, et al. Embodied agent interface: Benchmarking llms for embodied decision making. *Advances in Neural Information Processing Systems*, 37:100428–100534, 2024.

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.

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.

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.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

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.

Changyuan Wang, Chubin Zhang, Zhenyu Wu, Runhao Li, Angyuan Ma, Ke Chao, Yinan Liang, Xiuwei Xu, Ziwei Wang, Yansong Tang, et al. Skillmemo: Expert-guided skill memory framework for compositional embodied manipulation. *arXiv preprint arXiv:2608.05970*, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, et al. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL `https://arxiv.org/abs/2305.16291`.

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.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, et al. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. In *International Conference on Machine Learning*, pp. 70576–70631. PMLR, 2025.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026a.

Junyi Zhang, Jiaxin Ge, Hanjun Yoo, Letian Fu, Zihan Yang, Yaowei Liu, Raj Saravanan, Shaofeng Yin, Justin Yu, Dantong Niu, et al. Playful agentic robot learning. *arXiv preprint arXiv:2606.19419*, 2026b.

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*, 2026c.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

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.

A Appendix

A.1 EmbodiedBench Evolution Case Studies

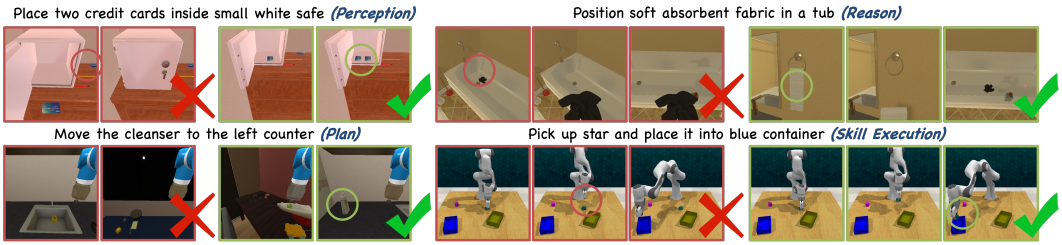

Figure 7: **Representative EmbodiedBench cases (GPT-5.5) before and after RoboFoundry evolution.** Examples illustrate repairs to perception, reasoning, planning, and low-level skill execution.

This section provides several examples of how RoboFoundry evolves on EmbodiedBench. Figure 7 illustrates four representative failures repaired through changes to perception, reasoning, planning, and skill execution. We compare the initial execution (GPT-5.5) with the final candidate promoted by RoboFoundry's evolution. Specifically, we expose three levels of change: (1) the persistent *general* system, (2) its *task-specific* realization during execution, and (3) the corresponding changes in the filesystem. To ensure a fair comparison, we use the same episode and instruction for the initial and final candidates.

A.1.1 Example 1: Feedback-Grounded Recovery in EB-Habitat

Consider the instruction "`Relocate every cleanser from sink onto the sofa.`" The initial system permits relatively long action sequences. The accepted revision instead introduces observation checkpoints after navigation and treats environment feedback as stronger evidence than visual appearance for interaction feasibility:

```
- Each plan should include no more than 20 actions.
+ Prefer one action; at most two when both are currently supported.
+ After Navigation, observe before Pick/Place/Open/Close.
+ Trust environment feedback over visual appearance for proximity.
+ After a not-near Pick failure, navigate before retrying.
```

This is a shared cross-task modification: it does not encode the cleanser, sink, sofa, or other entities from the proposal episode. During execution, the shared rule is instantiated using task-conditioned feedback. The persistent system policy stores the reusable recovery rule, while the runtime instantiation binds it to the current failed action, object, and environment feedback:

```
- Pick(cleanser) -> not near -> repeated retries -> failure
+ Pick(cleanser) -> not near
+ Navigate(table 2) -> Pick(cleanser) -> Navigate(sofa) -> Place(sofa)
+ task success
```

A.1.2 Example 2: Skill Refinement in EB-Manipulation

We further examine "`Please put the red star into the shape sorter.`" The initial rollout executes 15 valid actions without completing the task, whereas the rollout under the accepted revision succeeds at the sixth action. The revision introduces an explicit skill-level insertion strategy:

```
+ For shape-sorter tasks, insert the object into the matching opening.
+ Keep holding while aligning and descending into the opening.
+ Release only when the object is inside or just below the opening.
```

The corresponding task-conditioned action sequence changes from an early release to a grasp-preserving descent. Actions follow `[x,y,z,rx,ry,rz,gripper]`, where the final dimension controls the gripper:

```
- [49,60,48,0,60,45,0] // above opening, closed
- [49,60,40,0,60,45,1] // early release
+ [49,60,48,0,60,35,0] // above opening, closed
+ [49,60,39,0,60,35,0] // descend while holding
+ [49,60,38,0,60,35,1] // low-position release
```

A.1.3 Example 3: Support-Surface Grounding in EB-ALFRED

The instruction "`Place the newspaper next to the left laptop`" requires distinguishing a semantic spatial reference from a physically valid support surface. The laptop defines the target spatial region, while the sofa provides the executable support surface. The accepted revision explicitly separates these roles in the runtime plan:

```
find Newspaper
pick up Newspaper
find Laptop
+ find Sofa
put down
```

A.1.4 Example 4: Reachable-Floor Reasoning in EB-Navigation

For "`navigate to the Toaster and be as close as possible`", the initial planner primarily reasons about the target location and nearby obstacles. The planner under the accepted revision also uses shorter movement segments, allowing new observations and collision feedback to affect subsequent actions. The final target distance decreases from $3.9\,\mathrm{m}$ to $0.9\,\mathrm{m}$. The accepted revision adds an explicit rule for reasoning over reachable floor:

```
- Reason based on the target object's location and nearby obstacles.
+ Reason about the reachable floor path, not only the target image location.
+ For targets on furniture, approach the free floor beside the support surface.
```

A.1.5 Persistent Filesystem Evolution

RoboFoundry does not treat evolution as transient model output. A candidate becomes an *accepted revision* only after evaluation; accepted revisions are then committed to the agent workspace. For EmbodiedBench, the execution and outer-loop systems are organized as follows:

```
workspace/
+-- overlay/embodiedbench/
| +-- evaluator/config/
| | +-- system_prompts.py
| | +-- habitat_examples.json
| | +-- eb_navigation_examples.json
| | +-- eb_manipulation_examples.json
| | +-- eb_alfred_examples.json
| +-- planner/
| +-- vlm_planner.py
| +-- nav_planner.py
| +-- manip_planner.py
+-- .robofoundry/
+-- AGENTS.md
+-- READ_POLICY.md
+-- bootstrap/{snapshot.json, summary.md}
+-- experience/{parent_summary.json, parent/...}
+-- <benchmark>_feedback/{summary.md, failures.json}
```

Execution surface. `overlay/embodiedbench/` contains components directly used during agent execution. Evolution modifies persistent rules, demonstrations, and execution feedback. These modifications constitute the persistent system policy reused by subsequent executions. Representative file modifications are:

```
ALFRED
M evaluator/config/system_prompts.py
M planner/planner_utils.py
M planner/vlm_planner.py
Habitat
M evaluator/config/system_prompts.py
M evaluator/config/habitat_examples.json
```

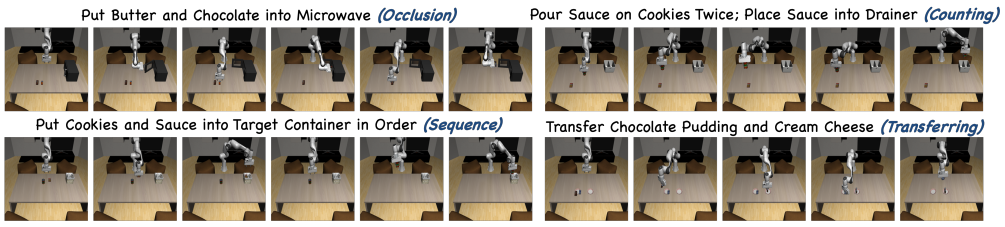

Figure 8: RoboFoundry rollouts across four memory categories in RoboMemArena: Transfer, Occlusion, Counting, and Sequence.

```
M planner/vlm_planner.py
Navigation
M evaluator/config/system_prompts.py
M evaluator/config/eb_navigation_examples.json
M planner/nav_planner.py
Manipulation
M evaluator/config/system_prompts.py
M evaluator/config/eb_manipulation_examples.json
M planner/manip_planner.py
```

Outer-loop evolution evidence. The `.robofoundry/` branch is the filesystem realization of the Evolution Ledger $\mathcal{L}$. It stores information used by the outer evolution process, including bootstrap state, parent–candidate information, evaluation summaries, and benchmark-specific failures. Observations, actions, feedback, and model outputs remain associated with evaluation traces:

```
evaluation/
+-- results/episode_*_res.json
+-- episode_*_step_*.json
+-- traces/episode_*/
    +-- step_*/
        +-- prompt.txt
        +-- model_output.txt
```

These execution traces provide the grounded evidence from which candidate system revisions are diagnosed and evaluated. Taken together, the filesystem separates two roles: runtime execution produces grounded traces, while accepted revisions persist reusable decision rules and execution constraints back into the system policy.

A.2 RoboMemArena Evolution Case Studies

We provide several examples of how RoboFoundry changes the context-management system on RoboMemArena. We expose how an accepted system revision changes (1) persistent context or memory rules, (2) the task-conditioned input, and (3) the corresponding files in the agent workspace. We select representative tasks from four RoboMemArena categories: Transfer, Occlusion, Counting, and Sequence. Figure 8 shows the corresponding rollouts and illustrates how the evolved context supports task execution.

A.2.1 Example 1: Task Progress as Active Context in Transfer

We first examine a Transferring task that requires moving both *butter* and *cream cheese* from one plate to another. The initial formatter ignores runtime progress and always returns the original task prompt, whereas the revised formatter explicitly selects the next unfinished stage:

```
- del context
- return str(base_prompt)
+ completed_count = _completed_stage_count(context)
+ stage = _next_stage_name(context)
+ overall = "overall task:  <base task>"
+ return <next stage> + <overall task> + "do not undo completed steps"
```

The persistent rule is task-agnostic: object names appear only in its runtime instantiation from the current execution state. For the selected episode:

```
Runtime state:
  01_Place_Butter_Plate2        = true
  02_Place_Cream_Cheese_Plate2 = false
```

This changes the actual input from a static task description to an explicit active subgoal:

```
- butter cream
+ place cream cheese plate 2.
+ overall task:  butter cream.
+ do not undo completed steps
```

The evolved context therefore preserves the active subgoal, global task, and completed progress simultaneously. The selected episode progresses from completing only the butter stage to completing both stages; across the corresponding 10-episode search evaluation, full-task success increases from $40\,\%$ to $70\,\%$.

A.2.2 Example 2: External Visual Memory under Occlusion

The Occlusion example requires manipulating cookies with a microwave and later placing popcorn into it. The evolved memory system introduces explicit rules for combining the current observation, sparse visual history, and attempted trials:

```
+ Ground decisions in the latest frame first.
+ Use historical keyframes for persistent objects, receptacles, and progress.
+ Keep frames that capture persistent state changes or useful locations.
+ Attempted commands are not evidence of physical success.
+ If the current image supports completion, advance to the next primitive.
```

The memory selector additionally provides sparse fallback anchors when no selected keyframes are available and summarizes recent attempted commands into the active context. For one selected invocation, the retrieved visual context is:

```
memory_indices_before = [0, 576, 672]
recent_window         = [711, ..., 715]
previous_primitive    = open microwave
place cookies         = attempted 31x (t=560--710)
```

The corresponding contextual guidance emphasizes that the retrieved historical frames are reminders rather than authoritative evidence:

```
+ Use sparse historical frames as reminders of persistent scene state.
+ Compare them with the current frame before deciding to continue or move on.
+ Attempted VLA prompts are not evidence that an action succeeded.
```

The initially selected episode repeatedly returns to `pick cookies`, whereas the evolved execution later transitions to `pick popcorn` and `place popcorn`. Recorded stage completion changes from $50\,\%$ to $75\,\%$. This example illustrates the evolution of both which historical observations are retrieved and how retrieved evidence is presented and interpreted.

A.2.3 Example 3: Counting as Stage-Conditioned Context

The Counting task requires two pouring operations followed by a final placement. The initial formatter always returns the same base prompt. The revised formatter instead converts ordered stage state into an explicit next-step instruction:

```
+ completed_count = number of completed stages
+ unfinished_count = number of remaining stages
+ append "<N> earlier steps already complete"
+ append "Next, <next unfinished stage>"
+ if one stage remains:  finish it and avoid redoing completed steps
```

Across the search evaluation, full-task success increases from $60\,\%$ to $80\,\%$. After both pouring stages have been recorded as complete, the runtime prompt changes to:

```
- pour tomato sauce into bowl drainer
+ Task:  pour tomato sauce into bowl drainer.
+ 2 earlier steps are already complete.
```

```
+ Next, place bowl drainer.
+ Complete this remaining step now and avoid redoing completed steps.
```

A.2.4 Example 4: Local Progress Improvement in Sequence

RoboFoundry changes the context system and selects key episodes. The evolved format exposes the active stage and adds stage-dependent execution constraints:

```
+ For place/put/insert/move:  the named item should end at the destination.
+ For open/close:  change only the named fixture for this step.
+ Do not redo completed steps.
+ After all stages complete:  preserve the final arrangement.
```

One runtime prompt becomes:

```
Current step: place cookies microwave.
Full task: pour tomato cookies microwave.
The named item should end at the named destination.
Do not redo completed steps.
```

After all listed stages are marked complete:

```
Full task: pour tomato cookies microwave.
All listed steps are complete;
hold the final arrangement and avoid disturbing objects.
```

A.2.5 Persistent Filesystem Changes

The behavior changes above correspond to persistent modifications in the execution workspace rather than transient natural-language suggestions:

```
candidates/<candidate>/workspace/overlay/
+-- evaluation_benchmark/
+-- scripts/
| +-- robofoundry_policy.py
+-- async_vlm_reference/
+-- memory_policy.py
+-- eval_async_vlm_vla.py
```

The stage-conditioned Transferring, Counting, and Sequence cases modify the shared context-management implementation, while the Occlusion case additionally changes visual-memory selection, retrieval, and active-context construction:

```
Transferring / Counting / Sequence
M evaluation_benchmark/scripts/robofoundry_policy.py
Occlusion
M evaluation_benchmark/async_vlm_reference/memory_policy.py
M evaluation_benchmark/async_vlm_reference/eval_async_vlm_vla.py
```

The persistent changes cover context formatting, ordered-stage processing, stage-conditioned context construction, replanning cadence, historical-frame selection, and memory-context formatting.

A.2.6 Evolution Evidence and Execution Traces

Outer-loop evolution evidence is stored separately from the execution-facing system. The `.robofoundry/` branch is the filesystem realization:

```
workspace/
+-- .robofoundry/
    +-- AGENTS.md
    +-- READ_POLICY.md
    +-- bootstrap/{snapshot.json, summary.md}
    +-- experience/
    |   +-- parent_summary.json
    |   +-- parent/{manifest.json, evaluation/, validation/}
    +-- <adapter>_feedback/
        +-- summary.md
        +-- failures.json
        +-- image_evidence/*.png
```

Task-conditioned evidence remains attached to each evaluation episode:

```
candidates/<candidate>/evaluation/<adapter>/run/taskN/epM/
+-- episode_result.json
+-- stage_events.jsonl
+-- vla_input_trace.jsonl       # runtime prompt / context
+-- vla_prompt_trace.jsonl      # VLA prompt
+-- sync_vlm_trace.jsonl        # VLM decision / memory indices
+-- vla_inputs/
+-- vlm_inputs/
```

This organization separates three roles: execution traces record episode-level state, actions, and failures; evolution evidence supports diagnosis and candidate evaluation; and accepted revisions persist reusable context-selection, retrieval, and utilization rules back into the system policy.

These cases expose complementary forms of context management. Transferring and Counting convert recorded progress into an explicit active subgoal; Occlusion changes the retrieval and utilization of external visual memory; and Sequence shows that better structured active context can improve intermediate progress without resolving the complete task. Together, these examples show that RoboFoundry evolves not only the content of a prompt, but also the mechanisms that determine what state is retained, what evidence is retrieved, and how it is presented to the embodied model.

A.3 LIBERO-PRO Evolution Case Studies

We further inspect how RoboFoundry revises the executable system on LIBERO-PRO. In contrast to the context-centric changes observed in RoboMemArena, the main revisions occur at the interface between high-level task reasoning and physical execution, i.e., code-as-policy. We distinguish two adaptation scopes. Persistent shared evolution modifies prompt templates, API semantics, reusable execution skills, and logic shared across tasks. In contrast, trial-local recovery occurs within an individual trial, where generated task code is revised using the current observation, simulator state, execution feedback, and previous failures.

A.3.1 Persistent Evolution

The selected persistent revision modifies four files in the shared workspace. These changes jointly modify task generation, API semantics, reusable execution skills, and environment launch:

```
candidates/<id>/workspace/
+-- env_configs/libero/
| +-- franka_libero_robofoundry.yaml
+-- robofoundry/integrations/franka/
| +-- libero_reduced.py
| +-- libero_harness.py
+-- robofoundry/utils/
+-- launch_utils.py
```

Task-generation prompt. The evolved template changes how ordinary object transfer should be implemented. The persistent rule specifies *how* a transfer should be executed without encoding object-specific cases:

```
- Prefer CuRobo grasp and transport planning with small fallbacks.
+ Prefer pick_and_place_sim_object(...)  for ordinary object transfer.
+ Pass the desired final object-center XYZ from the live target pose.
+ Use a modest positive Z clearance for receptacles or support surfaces.
+ Keep TCP offsets and bounded grasp retries inside the helper.
+ Use lower-level CuRobo only when scene geometry requires it.
```

Coordinate semantics. The revised execution interface consistently separates the desired object-center target from the TCP target used by the controller. This keeps hand-to-object offsets inside the embodiment-specific execution interface rather than repeatedly re-estimating them in generated task code:

```
object_position = get_sim_object_pose(object_name)[0]
tcp_position    = get_ee_pose()[0]
tcp_object_offset = tcp_position - object_position

tcp_target =
    desired_object_position + tcp_object_offset
```

Reusable helpers. The evolved system exposes three higher-level operations:

```
grasp_sim_object(...)
place_held_sim_object(...)
pick_and_place_sim_object(...)
```

The grasp skill reads the current object pose and performs bounded physical grasp attempts. The placement skill converts the requested object-center position into a TCP target and performs hover, descent, release, and retreat. The composite skill connects these operations. These skills execute physical motions rather than directly setting simulator object state.

A.3.2 Example 1: From Long Manipulation Code to Composite Skills

For task `"Pick the cream cheese and place it in the basket."`, the initial generated solution explicitly constructs segmentation, grasp generation, trajectory planning, lift validation, transport, descent, and release. Under the persistent shared revision, the generated task code instead binds live object poses to the reusable manipulation skills:

```
- segment cream cheese and generate grasp poses
- plan_grasp_trajectory(...)  and execute trajectory
- manually compute transport offset, descend, and release
+ cream_key = match cream cheese from live object poses
+ basket_pos = get_sim_object_pose(basket_key)[0]
+ target = basket_pos + [0, 0, 0.12]
+ pick_and_place_sim_object(cream_key, target, ...)
```

The persistent abstraction alone does not immediately complete the task. Furthermore, feedback-driven rewrites introduce a trial-local recovery. The composite manipulation skill is therefore persisted in the shared system, whereas the push–regrasp recovery remains local to this trial. This exposes two distinct adaptation time scales rather than a single monolithic code update:

```
read current cream-cheese and basket poses
        |
        v
construct a tangential push direction
        |
        v
push object and verify XY displacement
        |
        v
grasp_sim_object(...)
        |
        v
place_held_sim_object(...)
```

A.3.3 Example 2: Execution Abstraction without Spatial Grounding Success

For task instruction `"Pick the akita black bowl next to the ramekin and place it on the plate."`, both the initial and revised systems use simulator poses to identify the bowl nearest to the ramekin. The main persistent change is therefore the abstraction of the subsequent manipulation:

```
 select bowl nearest to ramekin using current XY distance
- segment bowl
- generate grasp poses
- plan_grasp_trajectory(...)
- manually lift, transport, descend, and release
+ target_pos = plate_pos + vertical clearance
+ pick_and_place_sim_object(bowl_key, target_pos, ...)
```

A.3.4 Filesystem Organization

The LIBERO-PRO workspace separates the persistent shared system, outer-loop evolution evidence, and trial-local generated task code.

Persistent executable system. The archived persistent revision modifies the task-generation interface, API semantics, reusable skills, and launch logic:

Table 5: **Main comparison on the EB-ALFRED high-level benchmark.** Avg. denotes the arithmetic mean over Base, Common Sense, Complex Instruction, Visual, Spatial, and Long Horizon. All results are reported in percentage (%).

Method	Avg.	Base	Common Sense	Complex Instruction	Visual	Spatial	Long Horizon
RoboFoundry (GPT-6 Astra)	**90.0**	**92.0**	**92.0**	**92.0**	**86.0**	**94.0**	84.0
RoboFoundry (GPT-5.5)	84.0	90.0	84.0	84.0	74.0	80.0	**92.0**
RoboFoundry (Qwen3.7-Plus)	81.3	88.0	90.0	72.0	74.0	82.0	82.0
RoboFoundry (GLM5.3-Flash)	80.0	82.0	76.0	82.0	80.0	74.0	86.0
RoboFoundry (Qwen3.8-27B)	78.0	82.0	84.0	82.0	72.0	68.0	80.0
RoboFoundry-Lite (GPT-5.5)	78.0	82.0	80.0	76.0	70.0	72.0	88.0
RoboFoundry-Lite (Qwen3.7-Plus)	75.0	76.0	80.0	82.0	74.0	62.0	76.0
GPT-6 Astra	87.3	**92.0**	**92.0**	**92.0**	80.0	88.0	80.0
GPT-5.5	76.7	86.0	82.0	82.0	70.0	72.0	68.0
Qwen3.7-Plus	73.3	82.0	78.0	66.0	78.0	56.0	80.0
GLM5.3-Flash	71.7	76.0	76.0	76.0	70.0	62.0	70.0
Claude-3.7-Sonnet	67.7	68.0	68.0	70.0	68.0	62.0	70.0
Qwen3.8-27B	65.7	72.0	76.0	60.0	62.0	68.0	56.0
Claude-3.5-Sonnet	64.0	72.0	66.0	76.0	60.0	58.0	52.0
GPT-4o	56.3	64.0	54.0	68.0	46.0	52.0	54.0

```
M env_configs/libero/franka_libero_robofoundry.yaml
M robofoundry/integrations/franka/libero_reduced.py
M robofoundry/integrations/franka/libero_harness.py
M robofoundry/utils/launch_utils.py
```

Outer evolution evidence. The `.robofoundry/` branch is the filesystem realization of the Evolution Ledger $\mathcal{L}$ and stores evidence used by the outer evolution process. The *curobo_debug* files are motion-planning debug artifacts:

```
workspace/
+-- .robofoundry/
|   +-- AGENTS.md
|   +-- bootstrap/{snapshot.json, summary.md}
|   +-- experience/
|   |   +-- parent_summary.json
|   |   +-- parent/{manifest.json, evaluation/, validation/}
|   +-- libero_pro_feedback/{report.md, summary.json}
+-- curobo_debug/
    +-- *.npz
    +-- *.obj
```

Task-specific execution. Generated task code and its feedback-driven revision history remain inside the corresponding evaluation trial:

```
candidates/<id>/evaluation/libero_pro/cells/<suite>/task_01/
+-- result.json
+-- stdout.txt
+-- stderr.txt
+-- config.yaml
+-- output/robofoundry/run/
    +-- trial_index.json
    +-- trial_01_.../
        +-- trial_manifest.json
        +-- code.py
        +-- summary.txt
        +-- motion_debug.json
        +-- codegen_trace/
            +-- initial/{prompt.txt, final/output.txt}
            +-- regen_*/{prompt.txt, final/output.txt}
```

The filesystem thus makes the two adaptation scopes explicit: persistent shared evolution stores reusable execution abstractions in the shared workspace, whereas trial-local recovery remains inside the corresponding trial and revises generated task code from execution feedback.

Table 6: **Main comparison on the EB-Habitat high-level benchmark.** Avg. denotes the arithmetic mean over Base, Common Sense, Complex Instruction, Visual, Spatial Relationship, and Long Horizon. All results are reported in percentage (%).

Method	Avg.	Base	Common Sense	Complex Instruction	Visual	Spatial	Long Horizon
RoboFoundry (GPT-5.5)	**88.7**	**100.0**	78.0	80.0	86.0	**100.0**	**88.0**
RoboFoundry (GPT-6 Astra)	86.7	**100.0**	**100.0**	**100.0**	**100.0**	42.0	78.0
RoboFoundry (Qwen3.7-Plus)	80.3	**100.0**	80.0	92.0	88.0	48.0	74.0
RoboFoundry (Qwen3.8-27B)	77.0	**100.0**	58.0	80.0	80.0	94.0	50.0
RoboFoundry (GLM5.3-Flash)	75.0	**100.0**	58.0	66.0	80.0	94.0	52.0
RoboFoundry-Lite (GPT-5.5)	74.3	**100.0**	66.0	74.0	80.0	44.0	82.0
RoboFoundry-Lite (Qwen3.7-Plus)	72.0	98.0	66.0	82.0	80.0	42.0	64.0
GPT-6 Astra	75.3	88.0	84.0	86.0	82.0	38.0	74.0
Claude-3.5-Sonnet	68.0	96.0	68.0	78.0	70.0	38.0	58.0
Qwen3.7-Plus	65.0	92.0	58.0	76.0	74.0	34.0	56.0
GPT-5.5	64.0	94.0	54.0	62.0	70.0	32.0	72.0
GLM5.3-Flash	60.7	94.0	54.0	54.0	64.0	34.0	64.0
GPT-4o	59.0	86.0	44.0	56.0	68.0	36.0	64.0
Claude-3.7-Sonnet	58.7	90.0	58.0	58.0	62.0	38.0	46.0
Qwen3.8-27B	58.0	96.0	46.0	48.0	60.0	36.0	62.0

Table 7: **Main comparison on the EB-Navigation benchmark.** Avg. denotes the arithmetic mean over Base, Common Sense, Complex Instruction, Visual, and Long Horizon. All results are reported in percentage (%).

Method	Avg.	Base	Common Sense	Complex Instruction	Visual	Long Horizon
RoboFoundry (GPT-6 Astra)	**80.3**	**90.0**	**85.0**	**86.7**	73.3	**66.7**
RoboFoundry (GLM5.3-Flash)	72.7	80.0	71.7	78.3	73.3	60.0
RoboFoundry (GPT-5.5)	72.0	80.0	83.3	80.0	76.7	40.0
RoboFoundry (Qwen3.7-Plus)	72.0	78.3	83.3	65.0	66.7	**66.7**
RoboFoundry-Lite (GPT-5.5)	69.7	83.3	81.7	76.7	**80.0**	26.7
RoboFoundry (Qwen3.8-27B)	68.3	76.7	**85.0**	73.3	63.3	43.3
RoboFoundry-Lite (Qwen3.7-Plus)	68.0	78.3	76.7	68.3	63.3	53.3
GPT-6 Astra	77.0	86.7	83.3	83.3	71.7	60.0
GLM5.3-Flash	71.0	80.0	73.3	80.0	66.7	55.0
Qwen3.7-Plus	64.3	75.0	73.3	63.3	60.0	50.0
GPT-4o	57.7	55.0	60.0	58.3	60.0	55.0
GPT-5.5	54.7	68.3	66.7	61.7	65.0	11.7
Qwen3.8-27B	52.3	41.7	66.7	60.0	53.3	40.0
Claude-3.7-Sonnet	45.0	50.0	61.7	50.0	36.7	26.7
Claude-3.5-Sonnet	44.7	66.7	51.7	41.7	36.7	26.7

Table 8: **Main comparison on the EB-Manipulation benchmark.** Avg. denotes the success rate over all 228 tasks. All results are reported in percentage (%).

Method	Avg.	Base	Common Sense	Complex Instruction	Visual	Spatial
RoboFoundry (GPT-6 Astra)	**54.8**	**58.3**	**62.5**	**56.3**	55.6	41.7
RoboFoundry (GLM5.3-Flash)	49.6	56.3	45.8	**56.3**	47.2	41.7
RoboFoundry (Qwen3.7-Plus)	47.8	43.8	54.2	52.1	44.4	43.8
RoboFoundry (GPT-5.5)	46.1	54.2	47.9	43.8	41.7	41.7
RoboFoundry (Qwen3.8-27B)	42.5	35.4	41.7	43.8	47.2	**45.8**
RoboFoundry-Lite (GPT-5.5)	39.5	43.8	37.5	31.3	47.2	39.6
RoboFoundry-Lite (Qwen3.7-Plus)	37.3	33.3	41.7	31.3	44.4	37.5
GPT-6 Astra	48.2	56.3	58.3	47.9	44.4	33.3
GLM5.3-Flash	43.0	41.7	41.7	33.3	**72.2**	33.3
Qwen3.7-Plus	36.0	35.4	35.4	27.1	61.1	27.1
Qwen3.8-27B	32.0	31.3	31.3	22.9	58.3	22.9
GPT-5.5	30.7	29.2	29.2	22.9	58.3	20.8
GPT-4o	28.9	39.6	29.2	29.2	19.4	25.0
Claude-3.7-Sonnet	28.5	31.3	20.8	43.8	25.0	20.8
Claude-3.5-Sonnet	25.4	37.5	16.7	29.2	19.4	22.9

Table 9: **Overall comparison on EmbodiedBench (16K).** Avg. is the arithmetic mean of success rates across the four suites. All results are reported in percentage (%). Gains are reported in percentage points (+pp).

Method	Avg.	EB-ALFRED	EB-Habitat	EB-Navigation	EB-Manipulation
RoboFoundry (GPT-5.5)	**71.7**$^{+8.3}$	82.3$^{+3.3}$	**81.3**$^{+12.3}$	75.7$^{+8.4}$	47.3$^{+9.0}$
RoboFoundry (Qwen3.7-Plus)	70.4$^{+5.8}$	**82.5**$^{+2.8}$	79.3$^{+5.6}$	71.8$^{+1.0}$	**47.8**$^{+13.7}$
RoboFoundry (Qwen3.8 27B)	69.4$^{+5.1}$	79.7$^{+3.4}$	79.3$^{+8.0}$	75.0$^{+6.0}$	43.7$^{+3.0}$
RoboFoundry (GLM5.3-Flash)	68.8$^{+7.6}$	80.0$^{+5.7}$	72.0$^{+5.3}$	**77.7**$^{+8.7}$	45.5$^{+10.5}$
Qwen3.7-Plus	64.6	79.7	73.7	70.8	34.1
Qwen3.8-27B	64.3	76.3	71.3	69.0	40.7
GPT-5.5	63.4	79.0	69.0	67.3	38.3
GLM5.3-Flash	61.3	74.3	66.7	69.0	35.0

Table 10: Task-wise LIBERO-PRO performance on the `libero-object` benchmark. Each entry reports position/task success rate in percentage (%). † denotes the use of privileged simulator object poses. Results for OpenVLA, π_0, $\pi_{0.5}$, and CaP-Agent0 are taken directly from the CaP-Agent0 report (Fu et al., 2026).

Task (Symbolic Form)	OpenVLA		π_0		$\pi_{0.5}$		CaP-Agent0		RoboFoundry		RoboFoundry†	
	Pos	Task	Pos	Task	Pos	Task	Pos	Task	Pos	Task	Pos	Task
Place(`alphabet_soup`, `basket`)	0.0	0.0	0.0	0.0	0.0	0.0	2.0	4.0	98.0	99.0	100.0	100.0
Place(`bbq_sauce`, `basket`)	0.0	0.0	0.0	0.0	100.0	2.0	12.0	42.0	97.0	93.0	100.0	100.0
Place(`butter`, `basket`)	0.0	0.0	0.0	0.0	54.0	0.0	26.0	18.0	97.0	99.0	100.0	100.0
Place(`chocolate_pudding`, `basket`)	0.0	0.0	0.0	0.0	0.0	2.0	18.0	48.0	88.0	93.0	100.0	100.0
Place(`cream_cheese`, `basket`)	0.0	0.0	10.0	0.0	0.0	0.0	12.0	6.0	98.0	99.0	100.0	100.0
Place(`ketchup`, `basket`)	0.0	0.0	0.0	0.0	20.0	2.0	32.0	12.0	96.0	100.0	90.0	100.0
Place(`milk`, `basket`)	0.0	0.0	0.0	0.0	0.0	0.0	38.0	2.0	98.0	98.0	100.0	100.0
Place(`orange_juice`, `basket`)	0.0	0.0	0.0	0.0	0.0	2.0	30.0	2.0	100.0	100.0	100.0	100.0
Place(`salad_dressing`, `basket`)	0.0	0.0	10.0	0.0	0.0	0.0	32.0	0.0	100.0	99.0	100.0	100.0
Place(`tomato_sauce`, `basket`)	0.0	0.0	0.0	0.0	0.0	0.0	16.0	48.0	88.0	100.0	100.0	100.0
Average	0.0	0.0	0.0	0.0	17.0	1.0	21.8	18.2	**96.0**	**98.0**	**99.0**	**100.0**

Table 11: Task-wise LIBERO-PRO performance on the `libero-goal` benchmark. Each entry reports position/task success rate in percentage (%). † denotes the use of privileged simulator object poses. Results for OpenVLA, π_0, $\pi_{0.5}$, and CaP-Agent0 are taken directly from the CaP-Agent0 report (Fu et al., 2026).

Task (Symbolic Form)	**OpenVLA**		π_0		$\pi_{0.5}$		**CaP-Agent0**		**RoboFoundry**		**RoboFoundry†**	
	Pos	Task	Pos	Task	Pos	Task	Pos	Task	Pos	Task	Pos	Task
Open(cabinet, drawer_mid)	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	71.0	71.0	95.0	100.0
Put(bowl, drawer_top)	0.0	0.0	0.0	0.0	94.0	2.0	4.0	0.0	90.0	71.0	95.0	100.0
Push(plate, stove_front)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	10.0	71.0	70.0	95.0	100.0
Put(bowl, plate)	0.0	0.0	0.0	0.0	0.0	2.0	36.0	38.0	90.0	97.0	100.0	100.0
Put(bowl, stove)	0.0	0.0	0.0	0.0	0.0	4.0	22.0	12.0	99.0	100.0	95.0	100.0
Put(bowl, cabinet_top)	0.0	0.0	0.0	0.0	0.0	2.0	60.0	4.0	100.0	99.0	99.0	100.0
Put(cream_cheese, bowl)	0.0	0.0	0.0	0.0	98.0	2.0	4.0	34.0	98.0	90.0	94.0	100.0
Put(wine_bottle, rack)	0.0	0.0	0.0	0.0	88.0	2.0	2.0	12.0	71.0	70.0	94.0	100.0
Put(wine_bottle, cabinet_top)	0.0	0.0	0.0	0.0	98.0	2.0	62.0	40.0	95.0	95.0	94.0	100.0
TurnOn(stove)	0.0	0.0	0.0	0.0	0.0	0.0	66.0	18.0	95.0	97.0	99.0	100.0
Average	0.0	0.0	0.0	0.0	38.0	0.0	25.6	16.8	**88.0**	**86.0**	**96.0**	**100.0**

Table 12: Task-wise LIBERO-PRO performance on the `libero-spatial` benchmark. Each entry reports position/task success rate in percentage (%). † denotes the use of privileged simulator object poses. Results for OpenVLA, π_0, $\pi_{0.5}$, and CaP-Agent0 are taken directly from the CaP-Agent0 report (Fu et al., 2026).

Task (Symbolic Form)	**OpenVLA**		π_0		$\pi_{0.5}$		**CaP-Agent0**		**RoboFoundry**		**RoboFoundry†**	
	Pos	Task	Pos	Task	Pos	Task	Pos	Task	Pos	Task	Pos	Task
Pick(between(plate, ramekin), plate)	0.0	0.0	0.0	0.0	2.0	0.0	22.0	14.0	98.0	88.0	100.0	100.0
Pick(table_center, plate)	0.0	0.0	0.0	0.0	0.0	2.0	22.0	14.0	89.0	99.0	100.0	100.0
Pick(drawer_top(cabinet_wood), plate)	0.0	0.0	0.0	0.0	0.0	0.0	2.0	10.0	88.0	96.0	99.0	98.0
Pick(next_to(cookie_box), plate)	0.0	0.0	0.0	0.0	0.0	2.0	0.0	10.0	97.0	96.0	100.0	98.0
Pick(next_to(plate), plate)	0.0	0.0	0.0	0.0	0.0	0.0	10.0	20.0	97.0	88.0	100.0	98.0
Pick(next_to(ramekin), plate)	0.0	0.0	0.0	0.0	12.0	2.0	30.0	14.0	89.0	96.0	85.0	100.0
Pick(on(cookie_box), plate)	0.0	0.0	0.0	0.0	0.0	0.0	14.0	8.0	88.0	88.0	98.0	100.0
Pick(on(ramekin), plate)	0.0	0.0	0.0	0.0	98.0	2.0	2.0	20.0	98.0	88.0	99.0	100.0
Pick(on(stove), plate)	0.0	0.0	0.0	0.0	2.0	0.0	8.0	14.0	88.0	88.0	100.0	100.0
Pick(on(cabinet_wood), plate)	0.0	0.0	0.0	0.0	90.0	0.0	8.0	16.0	88.0	88.0	99.0	98.0
Average	0.0	0.0	0.0	0.0	20.0	1.0	11.8	14.0	**92.0**	**91.5**	**98.0**	**99.2**