SCOPE: SELECTIVE CERTIFICATION OF PROGRAM EDITS FOR EMBODIED HEURISTIC LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Embodied heuristic learning must balance strategic adaptation with the preservation of previously successful behavior. Local repairs may leave structural failures unresolved, whereas broader revisions can introduce regressions that are costly to detect through repeated environment interaction. We introduce SCOPE (Selective Certification Of Program Edits), which uses failed and successful executions as complementary evidence for program revision. Frontier-guided program search annotates a shared diagnostic graph with execution evidence to distinguish supported progress from unresolved checks. Selective behavioral certification uses stateful API replay to restore recorded responses and mutations, checking whether edits preserve selected historical successes. Diagnosis and certification feedback, supplemented by sparse environment diagnostics, guide consecutive repairs across configurations. SCOPE outperforms the strongest baseline in both simulation and real-world tasks. On LIBERO-PRO and Robosuite, it achieves mean success rates of 80.66% and 84.48%, with respective gains of 6.51 and 3.48 percentage points.

1 INTRODUCTION

Robot control programs provide an explicit representation of how perception, decisions, and actions compose into task behavior. Language models can generate these programs from natural-language instructions, using control flow and API calls to express policies and situated task plans (Liang et al., 2023; Singh et al., 2023). Environment feedback then provides a basis for adapting behavior when execution departs from the plan (Huang et al., 2023b). This supports embodied heuristic learning, in which interaction improves task-specific decision rules while language-model parameters remain fixed. The resulting program must work across configurations, so learning requires more than solving the latest observed failure.

Execution-driven systems build on this approach through iterative program search. CaP-X studies coding agents that compose robot APIs and revise programs through interactive execution, while RHO optimizes policy repositories using execution feedback and validation-based candidate selection (Fu et al., 2026; Elmaaroufi et al., 2026). Reusable knowledge provides another source of improvement. Playful Agentic Robot Learning acquires code skills through self-directed exploration, and ASPIRE combines multimodal execution traces, validated repair knowledge, and evolutionary program search (Zhang et al., 2026; Lu et al., 2026). These approaches establish execution as both a source of learning evidence and a means of evaluating candidate programs. As revisions accumulate, the effects of each edit on previously successful behavior become an additional concern.

The difficulty is that a failure reveals where execution broke down, but the required change may extend beyond that point. A failed lift, for example, may originate in an unstable grasp, requiring the grasping stage to be redesigned rather than the lifting height to be adjusted. Local repairs can leave such structural failures unresolved, whereas broader revisions can alter decisions that supported success in other configurations. Success on the current configuration provides no direct evidence about those earlier behaviors. Repeating environment execution across historical successes can assess the change, but spends interaction budget on checking old behavior as well as discovering new behavior. Effective search therefore needs evidence to guide the extent of a repair and assess its consequences for prior successes.

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.

Our central insight is that failed and successful executions provide complementary evidence for program revision. Failed traces guide diagnosis and revision planning, while successful traces can be replayed to assess whether earlier behavior is preserved after an edit. By executing a revised program against the exchanges recorded in a successful trace, we can check whether it makes the same API requests and consumes the full trace with normal termination. Under faithful recording and deterministic execution, with all relevant external interactions captured at the API boundary, a complete match certifies success on the recorded instance. A divergence instead identifies where behavior changes and provides a target for further environment diagnosis. This turns historical interaction into feedback for subsequent program edits.

We introduce SCOPE (Selective Certification Of Program Edits), which couples frontier-guided program search with this selective certification feedback. SCOPE first constructs a shared diagnostic graph whose checks describe observable task progress and whose edges encode prerequisites. Execution evidence annotates the graph to identify a failure frontier consisting of unresolved checks with supported prerequisites. This frontier and its dependencies guide repair planning, allowing the model to select among local repair, stage redesign, and strategy regeneration. Revisions accumulate in a shared working program across development configurations.

SCOPE accepts the working program whenever it succeeds on a configuration. If this changes the accepted source, SCOPE samples historical successes for stateful full-trace API replay. Replay preserves recorded responses and public-state mutations, allowing later program decisions to depend on the same effects as in the original execution. For selected configurations that remain uncertified, sparse environment diagnostics provide fresh outcomes and failure evidence. These observations feed back into the same repair loop. Certification thus informs subsequent search through evidence of preserved behavior and changed interactions.

We evaluate the frozen programs on held-out configurations in LIBERO-PRO (Zhou et al., 2025) and Robosuite (Zhu et al., 2020). SCOPE achieves mean success rates of 80.66% and 84.48%, exceeding the strongest baseline on each benchmark by 6.51 and 3.48 percentage points, respectively. Component ablations support the roles of failure-frontier guidance, certification feedback, and explicit revision-level planning. Across the reported settings, full-trace replay takes 4.62–6.59% of the corresponding original environment execution time, demonstrating the low cost of obtaining behavioral evidence from recorded interactions.

Our contributions are:

- A program-search framework that uses execution evidence to localize failure frontiers and plan repairs at different revision levels, integrating historical-success feedback into consecutive repair.
- A selective certification mechanism based on stateful full-trace API replay, with an instance-specific success guarantee under the stated execution assumptions and sparse diagnostics for further repair.
- An evaluation on LIBERO-PRO and Robosuite showing improved mean held-out success over the strongest baseline on each benchmark, together with real-world experiments demonstrating strong performance on tasks requiring base–arm coordination, bimanual coordination, and fine manipulation.

2 PRELIMINARY

We study *embodied heuristic learning*, the refinement of task-specific decision rules through interaction. Control programs encode these heuristics through perception, conditional decisions, and actions. Given a task description d and public observation and control APIs, $P \in \mathcal{P}$ maps observation histories to API calls. An execution instance $\xi = (x_0, \omega, \kappa)$ specifies initial state and program inputs x_0, exogenous randomness ω, and API, runtime, and evaluator semantics κ. Let $Y(P;\xi) \in \{0,1\}$ indicate complete task success with normal termination. For the task distribution $\mathcal{D}_d$, we maximize

$$J(P) = \mathbb{E}_{\xi\sim\mathcal{D}_d}\big[Y(P;\xi)\big], \tag{1}$$

using finite development reset configurations $\mathcal{S}$ and per-configuration repair limits. Execution e on $s_e \in \mathcal{S}$ produces recorded interactions τ_e. Repeated resets have their own observations and

outcomes. Heuristic learning updates the program and memory while keeping language-model parameters fixed.

SCOPE couples *frontier-guided search* with *selective certification feedback*. The shared working program accumulates revisions across configurations. On each configuration, we execute it before repair, using observed failures to guide consecutive revisions and successful executions to update the accepted program. Replay and sparse diagnostics supply additional repair evidence. At search trial t, SCOPE maintains the state $\big(P_t^{\mathrm{acc}}, P_t^{\mathrm{work}}, G, \mathcal{A}_t, \mathcal{M}_t\big)$, comprising the accepted program, a working program shared across configurations, a diagnostic graph, an execution archive, and a historical success bank. Both programs start from a generated P_0. The archive and success bank are populated as development configurations are visited. Entry trials, repair candidates, and diagnostics are archived by role. Before a successful execution is available, repairs use the current failure record alone, allowing search to begin with $\mathcal{M}_0 = \varnothing$.

The replay analysis assumes that all behavior-relevant external inputs and effects cross the recorded API boundary, with program decisions and environment responses deterministic given ξ. Records faithfully preserve values, aliases, exceptions, and public-state mutations. Full matches cover all external events and normal termination. The fixed evaluator determines the outcome from the resulting state.

3 METHOD

SCOPE couples frontier-guided program search with selective certification feedback, as illustrated in Figure 1. Failed executions guide repairs, while historical successes provide evidence about the behavior of updated programs.

3.1 FAILURE-FRONTIER MEMORY AND RETRIEVAL

At the start of each task, a model plans a directed acyclic *diagnostic graph* $G = (V, E)$ from the task description, initial observations, and API specification. The graph is shared across development configurations. Nodes define observable checks, and edges encode prerequisites. Checks in $V_{\mathrm{inv}} \subseteq V$ require sustained support while dependent checks are established. The graph provides the reference for assessing execution traces and identifying supported progress and unresolved stages for repair.

A separate critic assesses each execution e against this graph, labeling each node $c_{e,v} \in \{\mathsf{S}, \mathsf{C}, \mathsf{U}\}$ (supported, contradicted, or unknown), citing a window $I_{e,v} = [\ell_{e,v}, u_{e,v}]$ in sampled public frames and API traces. Here, $\ell_{e,v}$ and $u_{e,v}$ denote the window's start and end execution steps, respectively. The host verifies frame provenance and cited-step endpoints, retaining raw judgments. In topological order, it retains $c_{e,v} = \mathsf{S}$ only when

$$\begin{aligned} &c_{e,u} = \mathsf{S} && \forall u \in \mathrm{pa}(v), \\ &u_{e,u} \le \ell_{e,v} && \forall u \in \mathrm{pa}(v) \setminus V_{\mathrm{inv}}, \\ &c_{e,u} = \mathsf{S}, \quad I_{e,v} \subseteq I_{e,u} && \forall u \in \mathrm{anc}(v) \cap V_{\mathrm{inv}}. \end{aligned} \tag{2}$$

Here $\mathrm{pa}(v)$ and $\mathrm{anc}(v)$ denote parents and ancestors. Ordinary prerequisites must precede the dependent window, while invariant ancestors must support its entire duration. Missing evidence or violated constraints change the label to U.

The supported nodes $S_e = \{v \in V : c_{e,v} = \mathsf{S}\}$ form the execution's supported prefix in G. The *failure frontier* contains unresolved checks with supported prerequisites:

$$F_e = \big\{v \in V \setminus S_e : \mathrm{pa}(v) \subseteq S_e\big\}. \tag{3}$$

The archive stores F_e with the source, raw execution, outcome, raw and filtered judgments, evidence windows, and failure hypotheses. Each execution thus yields an annotation of the shared graph, linking supported progress and unresolved stages to trace evidence. This guides the model in locating failures and inspecting their prerequisites. Records without usable assessments still supply raw observations, traces, and errors for repair.

For code generation, SCOPE retrieves the current failure record and, when available, the latest successful execution record associated with the accepted program P_t^{acc} from the task archive. The

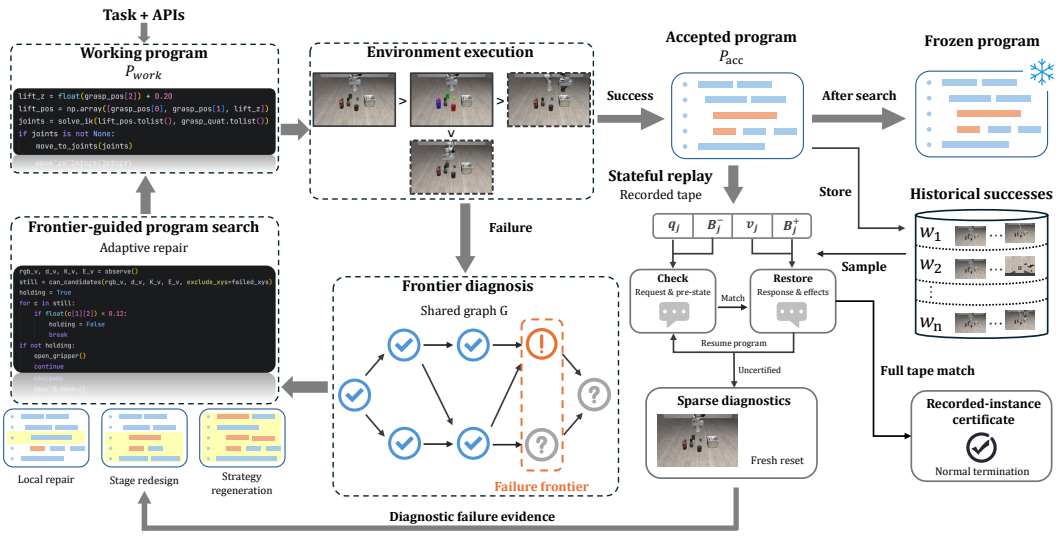


Figure 1: **Overview of SCOPE.** Execution evidence annotates a shared diagnostic graph to guide local repair, stage redesign, or strategy regeneration. Success updates the accepted program and historical success bank. When the accepted source changes, selected historical tapes are replayed by a local executor. Full matches certify the recorded instances under the replay assumptions. Sparse environment diagnostics on uncertified configurations supply failure evidence for subsequent repair. The accepted program is frozen after search.

failure supplies local diagnostic evidence, while the successful record provides a reference for behavior achieved by the accepted program. Before the first successful search trial, only the failure record is used. Each record retains its own source, observations, trace, outcome, and available diagnostic evidence. Edits use the shared working source, carrying accumulated revisions across configurations.

3.2 Adaptive Program Search

On its first main-search visit, each configuration is tested with the shared working program on a fresh reset. Success ends the visit. After failure, the critic annotates the trace against G, and the model revises the program using this evidence. The next execution on the same configuration validates the revision and, if it fails, supplies the diagnosis for another repair. Each configuration allows at most three main-search repairs. Replay feedback can trigger at most one additional follow-up repair per configuration.

To plan each repair, the model forms a diagnosis from the shared working source, the current execution's annotated graph, and the retrieved records. Existing full-trace feedback from successful witnesses and diagnostic outcomes also inform repairs (Section 3.3). Its repair plan states a failure hypothesis, cites execution evidence, identifies decisions and code regions to change, and describes the expected observable effect. Usable frontiers and dependencies help localize the cause. Otherwise, planning uses raw observations, traces, and errors. The model reports diagnosis, plan, and revision level in one planning request.

The model chooses among three revision levels. *Local repair* changes a localized condition, parameter, or code fragment. *Stage redesign* changes a task stage and, when needed, its prerequisites, while *strategy regeneration* reorganizes the overall control strategy. For example, evidence linking failed lifting to the grasp supports stage redesign. An edit's behavioral divergence probability bounds both potential gains and regressions (Appendix G). This motivates broader revisions when local repairs are insufficient, with execution evidence guiding the choice.

Local and stage repairs edit the shared working source, while regeneration may restructure it or restart from the accepted source. The model generates a complete candidate P' following the repair plan. The candidate becomes the shared working program for the next execution on that configura-

tion. The working source is otherwise unchanged. We archive the plan and revision level with the source and execution outcome to guide subsequent repairs.

The main queue visits development configurations in a fixed order. A FIFO queue holds diagnostic follow-ups. We alternate between nonempty queues at visit boundaries, completing consecutive repairs on a configuration before switching. Main-search and follow-up repairs are counted separately for each configuration, with counts retained on re-entry.

3.3 Success-Triggered Updates and Certification Feedback

At search trial t, the working program executes on a fresh reset of s_t, yielding $y_t = Y(P_t^{\mathrm{work}}; \xi_t)$. Every successful entry, candidate, or follow-up trial accepts the program that was executed:

$$A_t = y_t \mathbf{1}[P_t^{\mathrm{work}} \neq P_t^{\mathrm{acc}}],$$
$$P_{t+1}^{\mathrm{acc}} = \begin{cases} P_t^{\mathrm{work}}, & y_t = 1, \\ P_t^{\mathrm{acc}}, & y_t = 0. \end{cases} \tag{4}$$

Here A_t marks a change in the accepted source. Each successful search trial also updates the execution record associated with the accepted program for subsequent repair retrieval. All execution records remain archived. Full-trace checks and diagnostics follow accepted source updates, adding evidence for later repairs while leaving both programs unchanged.

The bank stores one *historical success witness* $w = (P_w, \xi_w, T_w)$ per retained configuration s_w, with $Y(P_w; \xi_w) = 1$. Every successful search trial adds or replaces the witness for s_t and retains others, including those with later diagnostic failures. Current-program outcomes are recorded separately.

SCOPE checks historical behavior through *stateful full-trace replay*, which restores recorded API effects to preserve the state read by subsequent program decisions. Figure 2 illustrates this mechanism with a grasp-and-place example. Each recorded execution e supplies a stateful API tape

$$T_e = \big((q_{e,j}, B_{e,j}^-, v_{e,j}, B_{e,j}^+)\big)_{j=1}^{n_e}, \tag{5}$$

where $q_{e,j}$ contains the API name and arguments, $v_{e,j}$ is the return value or exception, and $B_{e,j}^-, B_{e,j}^+$ are public object states before and after the call. A canonical encoding preserves values, shared references, array views, and API mutations to objects retained by the program. A local executor runs the tested program from its entry point, comparing each request and pre-call state with $(q_{e,j}, B_{e,j}^-)$. Matches apply recorded mutations and reproduce $v_{e,j}$. Divergence stops replay. A full-tape match also requires normal termination.

After an accepted source update ($A_t = 1$), we sample up to $K = 3$ success witnesses uniformly without replacement from $\mathcal{M}_t$. This gives a certification set $\mathcal{W}_t \subseteq \mathcal{M}_t$ that probes behavior preservation across the historical success bank. Let $C_t(P, w) = 1$ denote a full-tape match with normal termination, and $C_t(P, w) = 0$ otherwise. Under the assumptions in Section 2, $C_t(P, w) = 1$ certifies success on the recorded instance ξ_w. Selected but uncertified witnesses are

$$\mathcal{U}_t(P) = \{w \in \mathcal{W}_t : C_t(P, w) = 0\}. \tag{6}$$

Feedback distinguishes the first divergent interaction from unavailable replay. Divergence indicates changed behavior, not task failure. Each record identifies the tested program and witness so later repairs can locate affected decisions.

After each accepted source update ($A_t = 1$), *sparse environment diagnostics* execute the accepted program on at most k_{diag} configurations from $\mathcal{U}_t(P_t^{\mathrm{work}})$ in the sampled order. We select configurations whose follow-up repair remains unused and that have no pending follow-up. A failed diagnostic enqueues the configuration with source, trace, and available frontier evidence.

A follow-up repairs directly from the diagnostic failure if its source is still the shared working program, using one execution to validate the revision. Otherwise, it first executes the current program. Success ends the visit, while failure prompts one repair and its validating trial. Follow-ups use the same diagnosis, retrieval, and update rules as main search. Fresh resets describe their own instances, which may differ from ξ_w. Replay and diagnostic counts are bounded per accepted source update. Appendix H gives the certification proof and cost analysis. After completing search and servicing pending follow-ups, we freeze the accepted program.

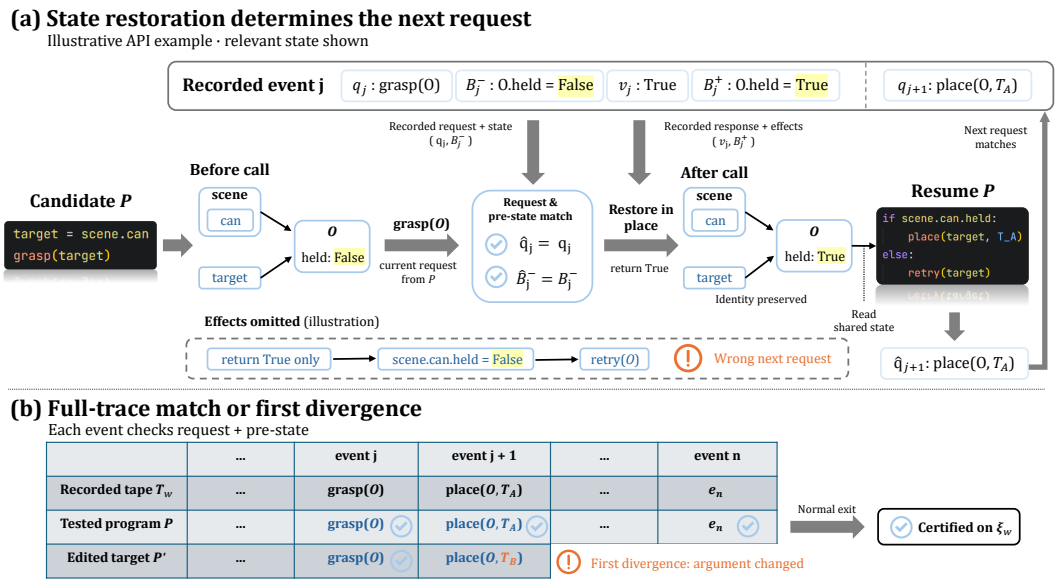


Figure 2: **Stateful full-trace replay. (a)** A local executor compares each API request and pre-call state with the tape, reproduces the recorded response and mutations while preserving shared references, and resumes the program. Omitting a mutation changes the next request in this illustrative example. **(b)** Matching every event, making no extra external calls, and terminating normally certifies success on the recorded instance ξ_w under the replay assumptions. A changed argument stops replay at the first divergence, leaving the instance uncertified.

4 Experiments

Our experiments address three questions: (1) Does SCOPE improve the generalization of learned control programs to held-out configurations? (2) How do certification feedback, failure-frontier guidance, and revision-level planning contribute to performance? (3) How often does replay certify historical successes, and what computational costs do certification and diagnostics incur? We also evaluate frozen programs on three physical manipulation tasks and examine how execution feedback changes a real-world control strategy.

4.1 Experimental Setup

We evaluate LIBERO-PRO (Zhou et al., 2025), an extension of LIBERO (Liu et al., 2023), and Robosuite (Zhu et al., 2020). LIBERO-PRO covers Object, Goal, and Spatial suites under position swaps (Pos) and task changes (Task), totaling 60 task–setting pairs. Robosuite covers seven manipulation tasks. Our primary comparison uses GLM-5.3-Flash for both SCOPE and ASPIRE (Lu et al., 2026), alongside other program-based and vision-language-action baselines.

Programs are generated, repaired, and selected on development configurations, then frozen for held-out evaluation without further repair. Our simulation results report task-averaged success as mean $\pm$ sample standard deviation over three independent learning and evaluation runs. For the other comparison methods, we report point estimates under their original protocols. Seed splits, implementation details, search budgets, and baseline sources appear in Appendix B.

4.2 Benchmark Results

On LIBERO-PRO, SCOPE achieves $80.66 \pm 1.46\%$ overall success (Table 1), improving over ASPIRE with the same GLM-5.3-Flash backbone by 6.51 percentage points and leading all six suite–setting combinations. The largest gains over this baseline occur on Goal–Task and Spatial–Pos, where success rises from 48.80% to 61.87% and from 54.47% to 65.40%, respectively. The

gains therefore extend beyond the Object suite, where both methods already exceed 95% in each setting. Individual task results appear in Appendix F.

Table 1: **LIBERO-PRO results.** Success rates (%) under position swaps (Pos) and task changes (Task). Overall averages over suites, and All averages over both settings.

Method	Object		Goal		Spatial		Overall		
	Pos	Task	Pos	Task	Pos	Task	Pos	Task	All
OpenVLA	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
π_0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
SUREFlow	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MolmoAct	6.00	0.00	0.00	0.00	0.00	0.00	2.00	0.00	1.00
X-VLA	2.00	8.00	1.00	9.00	0.00	0.00	1.00	5.67	3.33
$\pi_{0.5}$	17.00	1.00	38.00	0.00	20.00	1.00	25.00	0.67	12.83
CaP-Agent0	22.00	18.00	26.00	17.00	12.00	14.00	20.00	16.33	18.17
RATs	61.00	63.00	43.00	36.00	29.00	31.00	44.33	43.33	43.83
RHO	12.20	37.20	50.60	55.60	53.00	61.60	38.60	51.47	45.03
ASPIRE Opus 4.6	97.80	95.40	80.80	45.40	51.00	59.60	76.53	66.80	71.67
ASPIRE GLM-5.3-Flash	98.07 ± 0.50	95.60 ± 3.82	83.40 ± 2.16	48.80 ± 11.68	54.47 ± 3.45	64.53 ± 1.17	78.64 ± 1.74	69.64 ± 4.78	74.14 ± 3.11
SCOPE GLM-5.3-Flash	**99.40** ± 0.60	**97.73** ± 0.61	**88.33** ± 0.61	**61.87** ± 5.41	**65.40** ± 3.82	**71.20** ± 1.11	**84.38** ± 1.25	**76.93** ± 2.19	**80.66** ± 1.46

RHO and RATs use 50 and 10 evaluation trials per task, respectively.

On Robosuite, SCOPE attains $84.48 \pm 1.67\%$ mean success (Table 2), improving over ASPIRE with the same GLM-5.3-Flash backbone by 4.29 percentage points. It improves all seven task means in the same-backbone comparison. The largest gain is on two-arm lift, where success rises from 70.33% to 86.00%.

Table 2: **Robosuite results.** Success rates (%) on seven manipulation tasks. Mean averages over tasks.

Method	Cube lift	Cube stack	Cube restack	Spill wipe	Two arm handover	Two arm lift	Nut assembly	Mean
CaP-Agent0	97.00	98.00	89.00	**100.00**	20.00	74.00	0.00	68.29
RHO†	98.00	98.00	54.00	**100.00**	86.00	53.00	1.00	70.00
ASPIRE Opus 4.6	97.00	**99.00**	**100.00**	99.00	**92.00**	71.00	9.00	81.00
ASPIRE GLM-5.3-Flash	98.33 ± 1.53	97.33 ± 1.53	97.67 ± 1.53	98.33 ± 1.15	88.67 ± 5.13	70.33 ± 6.81	10.67 ± 4.73	80.19 ± 2.14
SCOPE GLM-5.3-Flash	**100.00** ± 0.00	**99.00** ± 1.00	**100.00** ± 0.00	**100.00** ± 0.00	91.67 ± 3.21	**86.00** ± 3.61	**14.67** ± 4.93	**84.48** ± 1.67

† RHO's 100 evaluation trials per task include 10 development trial IDs.

On the Quanta X2 robot from X Square Robot, we test base–arm coordination (*Cube into basket*), bimanual coordination (*Two arm lift*), and fine manipulation (*Cube restack*). Both methods use ten development placements and share the language model, perception modules, and control interfaces. Each frozen program is tested on 20 held-out placements. Hardware details, search budgets, and task sequences appear in Appendix E and Figure 5. SCOPE achieves 66.7% overall success versus ASPIRE's 40.0%, improving by 30, 20, and 30 percentage points on the three tasks, respectively (Table 3). Cube restack remains the hardest task for both methods, at 55.0% and 25.0% success. During development, stateful full-trace replay checks recorded behavior locally without requiring exact physical reconstruction of earlier scenes.

Figure 3 presents a qualitative case of program repair on a physical robot tasked with placing a cube into a basket. The gripper initially occludes the cube, preventing localization. Moving backward

Table 3: **Real-world generalization.** Successes/trials (success rate) for frozen programs on 20 held-out physical configurations per task. Overall pools 60 trials per method. Bold marks the higher success rate.

Task	Capability	ASPIRE	SCOPE
Cube into basket	Base–arm coordination	9/20 (45.0%)	**15/20 (75.0%)**
Two arm lift	Bimanual coordination	10/20 (50.0%)	**14/20 (70.0%)**
Cube restack	Fine manipulation	5/20 (25.0%)	**11/20 (55.0%)**
Overall	—	24/60 (40.0%)	**40/60 (66.7%)**

reveals the target but leaves it out of reach. Moving forward restores reachability while introducing occlusion again. The agent then revises the program to cache the target position before approaching, enabling grasping and placement to proceed.

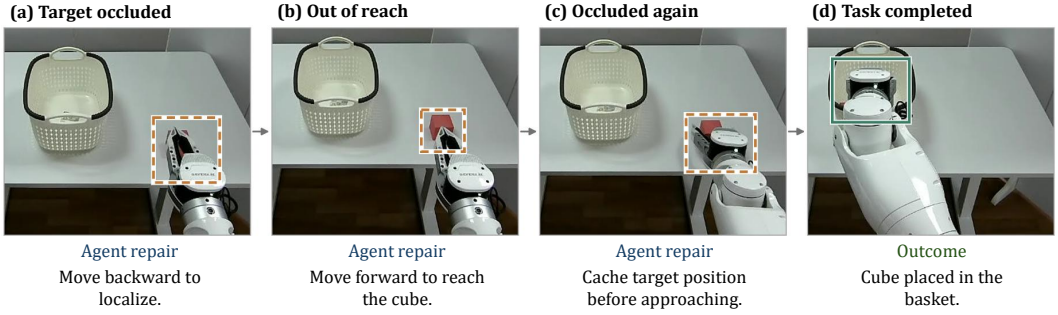


Figure 3: **Real-world repair of a cube-in-basket task. (a)** Gripper occlusion prompts backward motion for localization. **(b)** The target is visible but out of reach, prompting forward motion. **(c)** Renewed occlusion motivates caching the target position before approaching. **(d)** The revised program completes the task. Each panel pairs a key frame with the corresponding repair or outcome.

4.3 Component Ablations

We separately mask certification feedback, mask failure-frontier guidance, or omit explicit revision-level selection on all ten tasks in each of Goal–Task and Spatial–Pos. Variants share per-configuration repair limits and evaluation protocols. Feedback masking leaves internal replay and environment diagnostics active. Appendix C specifies the controls and retained information.

Table 4: **Component ablations on LIBERO-PRO.** Success rates (%) on all 10 tasks in each setting with GLM-5.3-Flash, reported as mean ± sample standard deviation over three runs.

Method / variant	Goal–Task	Spatial–Pos
ASPIRE (baseline)	48.80 ± 11.68	54.47 ± 3.45
Full SCOPE	61.87 ± 5.41	65.40 ± 3.82
w/o certification feedback	57.60 ± 7.33	59.33 ± 4.16
w/o failure-frontier guidance	55.07 ± 9.10	59.93 ± 3.72
w/o explicit revision-level planning	58.87 ± 7.11	62.13 ± 3.88

All three removals lower mean success in both settings. Removing failure-frontier guidance causes the largest decrease on Goal–Task (6.80 percentage points), while removing certification feedback causes the largest decrease on Spatial–Pos (6.07 points). These results support combining failure localization, behavior-preservation feedback, and explicit revision-level planning.

4.4 Certification Rates and Costs

We measure how often replay certifies historical successes and the time required to obtain this feedback (Figure 4). Table 5 in Appendix D provides the full numerical results. Certification rates count full matches when checking older success witnesses under a different program, with unsupported checks retained in the denominator. All quantities are task macro-averages. Full matches account for 38.9–60.6% of these checks across LIBERO-PRO settings and 44.7% on Robosuite. The remaining checks identify behavior changes or unavailable replay evidence. Sparse environment diagnostics assess the current program on selected unresolved configurations.

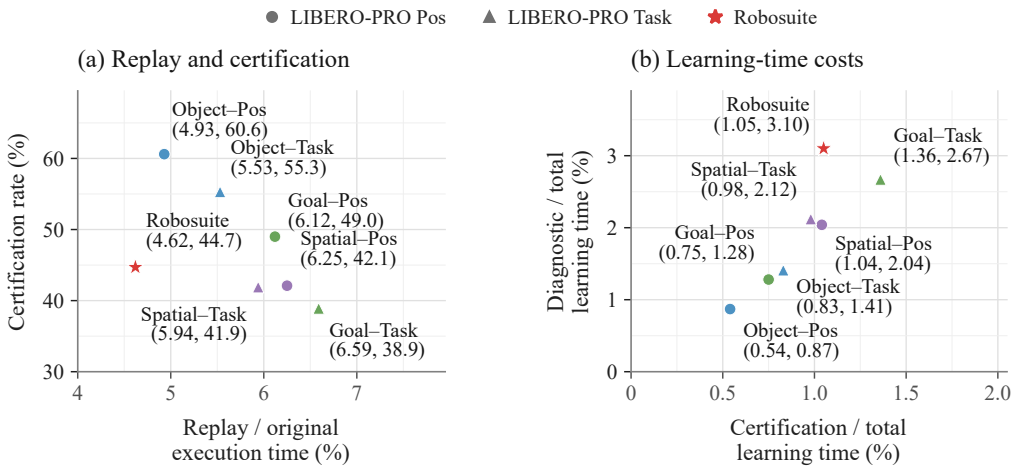


Figure 4: **Certification rates and feedback costs of SCOPE.** Each point is a task macro-average for one setting with GLM-5.3-Flash, covering 60 LIBERO-PRO task–setting pairs and seven Robosuite tasks. **(a)** Certification rate versus replay time relative to the corresponding original environment execution. **(b)** Certification-pipeline and diagnostic environment-execution times, each relative to total development learning time. Labels give (x, y) in percent. Circles and triangles denote LIBERO-PRO Pos and Task settings, while stars denote Robosuite. The certification pipeline includes replay, retrieval, verification, and self-matching. Diagnostic time covers environment execution only.

Replay takes 4.62–6.59% of the corresponding original environment execution time across the reported settings (Figure 4a). The full-trace certification pipeline, including retrieval, verification, and self-matching, accounts for 0.54–1.36% of total development learning time. Diagnostic environment executions account for a further 0.87–3.10% of development time (Figure 4b). These measurements show that recorded executions can provide behavioral evidence at a small fraction of their original execution cost, while sparse diagnostics supply fresh observations for further repair.

5 Conclusion and Limitations

We presented SCOPE, which uses failed and successful executions as complementary evidence for embodied heuristic learning. Prerequisite-aware diagnosis guides revisions, while selective replay assesses how accepted edits affect recorded successful behavior. This feedback loop supports adaptation across configurations without updating language-model parameters. On LIBERO-PRO and Robosuite, SCOPE achieves mean held-out success rates of 80.66% and 84.48%, exceeding the strongest baseline on each benchmark by 6.51 and 3.48 percentage points, respectively. Full-trace replay takes 4.62–6.59% of the corresponding original environment execution time, making historical behavior inexpensive to assess during search.

Repeated language-model inference remains costly. Recorded token usage implies an estimated cost of several thousand US dollars for running the full experimental suite through hosted model APIs. On physical robots, early candidate programs may produce unsafe behavior, requiring human supervision and intervention. These limitations motivate future work on more selective model querying and runtime safety constraints to support efficient program learning on physical robots.

REFERENCES

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris Coll-Vinent, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, Jie Lei, Tengyu Ma, Baishan Guo, Arpit Kalla, Markus Marks, Joseph Greer, Meng Wang, Peize Sun, Roman Rädle, Triantafyllos Afouras, Effrosyni Mavroudi, Katherine Xu, Tsung-Han Wu, Yu Zhou, Liliane Momeni, Rishi Hazra, Shuangrui Ding, Sagar Vaze, Francois Porcher, Feng Li, Siyuan Li, Aishwarya Kamath, Ho Kei Cheng, Piotr Dollar, Nikhila Ravi, Kate Saenko, Pengchuan Zhang, and Christoph Feichtenhofer. SAM 3: Segment anything with concepts. In *International Conference on Learning Representations*, 2026.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. CodeT: Code generation with generated tests. In *International Conference on Learning Representations*, 2023.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, 2024.

Yuhui Chen, Shuai Tian, Shugao Liu, Yingting Zhou, Haoran Li, and Dongbin Zhao. ConRFT: A reinforced fine-tuning method for VLA models via consistency policy. In *Proceedings of Robotics: Science and Systems*, 2025.

Christopher Clark, Jieyu Zhang, Zixian Ma, Jae Sung Park, Mohammadreza Salehi, Rohun Tripathi, Sangho Lee, Zhongzheng Ren, Chris Dongjoo Kim, Yinuo Yang, Vincent Shao, Yue Yang, Weikai Huang, Ziqi Gao, Taira Anderson, Jianrui Zhang, Jitesh Jain, George Stoica, Winson Han, Ali Farhadi, and Ranjay Krishna. Molmo2: Open weights and data for vision-language models with video understanding and grounding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-E: An embodied multimodal language model. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Karim Elmaaroufi, Justin Svegliato, Sarunas Kalade, Graham Schelle, Sanjit A. Seshia, and Matei Zaharia. RHO: Your coding agent is secretly a roboticist. *arXiv preprint arXiv:2606.16458*, 2026.

Hao-Shu Fang, Chenxi Wang, Minghao Gou, and Cewu Lu. GraspNet-1Billion: A large-scale benchmark for general object grasping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi Jim Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026.

Jiaheng Hu, Rose Hendrix, Ali Farhadi, Aniruddha Kembhavi, Roberto Martín-Martín, Peter Stone, Kuo-Hao Zeng, and Kiana Ehsani. FLaRe: Achieving masterful and adaptive robot policies with large-scale reinforcement learning fine-tuning. In *IEEE International Conference on Robotics and Automation*, 2025.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. VoxPoser: Composable 3D value maps for robotic manipulation with language models. In *Proceedings of The 7th Conference on Robot Learning*, 2023a.

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 *Proceedings of The 6th Conference on Robot Learning*, 2023b.

Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. ReKep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. In *Proceedings of The 8th Conference on Robot Learning*, 2025.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as i can, not as i say: Grounding language in robotic affordances. In *Proceedings of The 6th Conference on Robot Learning*, 2023.

Md Tanvir Islam, Sai Navaneet Peddapalli, Sangmoon Lee, and Sangtae Ahn. SUREFlow: State-space uncertainty-aware REsidual flow matching for robust robot manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2026.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Proceedings of The 2nd Conference on Robot Learning*, 2018.

Jason Lee, Jiafei Duan, Haoquan Fang, Yuquan Deng, Shuo Liu, Boyang Li, Bohan Fang, Jieyu Zhang, Yi Ru Wang, Sangho Lee, Winson Han, Wilbert Pumacay, Angelica Wu, Rose Hendrix, Karen Farley, Eli VanderBilt, Ali Farhadi, Dieter Fox, and Ranjay Krishna. MolmoAct: Action reasoning models that can reason in space. In *IEEE International Conference on Robotics and Automation*, 2026.

Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Zhaohui Yang, Kaiyan Zhang, Xuekai Zhu, Yuchen Zhang, Tianxing Chen, Ganqu Cui, Dehui Wang, Dingxiang Luo, Yuchen Fan, Youbang Sun, Jia Zeng, Jiangmiao Pang, Shanghang Zhang, Yu Wang, Yao Mu, Bowen Zhou, and Ning Ding. SimpleVLA-RL: Scaling VLA training via reinforcement learning. In *International Conference on Learning Representations*, 2026.

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.

LIBERO-PRO. LIBERO-PRO model leaderboard, 2026.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023.

Jijia Liu, Feng Gao, Bingwen Wei, Xinlei Chen, Qingmin Liao, Yi Wu, Chao Yu, and Yu Wang. What can RL bring to VLA generalization? an empirical study. In *Advances in Neural Information Processing Systems*, 2025.

Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, Mosharaf Chowdhury, Yuke Zhu, Linxi Jim Fan, and Guanzhi Wang. ASPIRE: Agentic /Skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026.

Jianlan Luo, Charles Xu, Jeffrey Wu, and Sergey Levine. Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning. *Science Robotics*, 2025.

Yecheng Jason Ma, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. LIV: Language-image representations and rewards for robotic control. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Jim Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-Tau Yih, Sida Wang, and Xi Victoria Lin. LEVER: Learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Allen Z. Ren, Justin Lidard, Lars L. Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. In *International Conference on Learning Representations*, 2025.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Junyang Shu, Zhiwei Lin, and Yongtao Wang. RFTF: Reinforcement fine-tuning for vision-language-action models with temporal feedback. In *ICLR Workshop on AI with Recursive Self-Improvement*, 2026a.

Junyang Shu, Zhiwei Lin, Bingqing Wei, and Yongtao Wang. Feat2Go: Visual feature-grounded value estimation for embodied reinforcement learning. *arXiv preprint arXiv:2605.30795*, 2026b.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. In *IEEE International Conference on Robotics and Automation*, 2023.

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. *Transactions on Machine Learning Research*, 2024.

X Square Robot. Quanta X2, n.d.a. URL `https://x2robot.com/en/product/quantum2`. Official product specifications. Accessed September 25, 2026.

X Square Robot. API Documentation – Quanta_X2, n.d.b. URL `https://github.com/X-Square-Robot/sdk_robot/blob/main/docs/API_Quanta_X2.md`. Official robot SDK documentation. Accessed September 25, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Takuma Yoneda, Jiading Fang, Peng Li, Huanyu Zhang, Tianchong Jiang, Shengjie Lin, Ben Picker, David Yunis, Hongyuan Mei, and Matthew R. Walter. Statler: State-maintaining language models for embodied reasoning. In *IEEE International Conference on Robotics and Automation*, 2024.

Junyi Zhang, Jiaxin Ge, Hanjun Yoo, Letian Fu, Zihan Yang, Yaowei Liu, Raj Saravanan, Shaofeng Yin, Justin Yu, Dantong Niu, Zirui Wang, Roei Herzig, Ken Goldberg, Yutong Bai, David M. Chan, Ion Stoica, Angjoo Kanazawa, Jiahui Lei, Haiwen Feng, and Trevor Darrell. Playful agentic robot learning. *arXiv preprint arXiv:2606.19419*, 2026.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, Tai Wang, Ya-Qin Zhang, Jingjing Liu, and Xianyuan Zhan. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *International Conference on Learning Representations*, 2026.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, 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.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Kevin Lin, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

A Related Work

Embodied control and skill learning. Language models connect task instructions to embodied action through grounded planning, multimodal representations, and structured control interfaces (Ichter et al., 2023; Driess et al., 2023; Huang et al., 2025). Code as Policies and ProgPrompt compose robot APIs into programs with control flow and task preconditions (Liang et al., 2023; Singh et al., 2023), while VoxPoser constructs 3D value maps that express spatial affordances and constraints for motion planning (Huang et al., 2023a). Environmental feedback and explicit world-state memory support adaptation during execution in Inner Monologue and Statler (Huang et al., 2023b; Yoneda et al., 2024). Learning also extends to reusable skills. Voyager builds a library of executable programs through an automatic curriculum in Minecraft, and Playful Agentic Robot Learning acquires code skills through self-directed robot exploration for subsequent tasks (Wang et al., 2024; Zhang et al., 2026). These representations support instruction grounding, state tracking, and skill reuse. SCOPE makes task progress explicit in a diagnostic graph shared across configurations. Prerequisite links connect execution evidence to unresolved stages, allowing repairs to target causes beyond the visible failure and select the revision level accordingly.

Embodied reinforcement learning. Embodied reinforcement learning and SCOPE share an iterative feedback loop for improving behavior. An agent executes its current policy, uses environment feedback to revise its behavior, and evaluates the update through further interaction. QT-Opt learns visual grasping through scalable off-policy Q-learning (Kalashnikov et al., 2018), while HIL-SERL combines demonstrations and corrective human interventions with online reinforcement learning for precise manipulation (Luo et al., 2025). Reinforcement fine-tuning further adapts demonstration-trained policies through interaction, including diffusion policies and vision-language-action models (Hu et al., 2025; Ren et al., 2025; Chen et al., 2025; Li et al., 2026). Controlled evaluations show that reinforcement fine-tuning can improve generalization under semantic and execution shifts (Liu et al., 2025). Reward synthesis and learned estimates of task progress enrich feedback for policy learning and planning (Ma et al., 2023; 2024; Shu et al., 2026a;b). SCOPE implements this loop through successive control-program revisions, carrying learned decision rules across trials while keeping language-model parameters fixed. Its prerequisite-aware diagnoses link feedback to local repairs, stage redesign, or strategy regeneration, enabling targeted adaptation without policy-parameter training. Under the assumptions in Section 2, full-trace replay also certifies success on matched recorded instances, making historical interaction available as evidence of preserved behavior after an edit.

Feedback-driven program repair and search. When control is expressed as executable programs, feedback can guide code revisions and the search over candidate policies. Environment interaction, self-reflection, and tree search offer complementary ways for language agents to revise their decisions (Yao et al., 2023; Shinn et al., 2023; Zhou et al., 2024). Self-Refine iterates between self-feedback and revision, and Self-Debugging combines program execution with natural-language explanations to repair code (Madaan et al., 2023; Chen et al., 2024). Executable evaluation supports testing, candidate selection, and population search over programs (Chen et al., 2023; Ni et al., 2023; Romera-Paredes et al., 2024). For robot control, CaP-Agent0 combines multi-turn visual feedback, synthesized utilities, and parallel reasoning within CaP-X (Fu et al., 2026). ASPIRE is particularly relevant because it integrates per-primitive multimodal traces, validated repair knowledge, and evolutionary search, evaluating candidate programs through environment execution (Lu et al., 2026). RHO optimizes policy repositories using paired candidate evaluation and a frontier of programs that maintains coverage of validation instances (Elmaaroufi et al., 2026). SCOPE augments this search with historical-success feedback after accepted source updates. Stateful API replay identifies preserved traces and the first divergent interactions, while sparse environment diagnostics provide fresh evidence for repair. Subsequent revisions can therefore account for an edit's effects on earlier successes as well as the current failure, without repeating environment execution for certified traces.

B EXPERIMENTAL SETUP DETAILS

Benchmarks and evaluation protocol. We use LIBERO-PRO (Zhou et al., 2025), an extension of LIBERO (Liu et al., 2023), and Robosuite (Zhu et al., 2020). Following ASPIRE (Lu et al., 2026), we evaluate the Object, Goal, and Spatial suites under position swaps (Pos) and task changes (Task). Each suite contains ten tasks, yielding 60 task–setting pairs. We also evaluate seven Robosuite manipulation tasks, including object lifting, stacking, wiping, bimanual manipulation, and nut assembly. We retain ASPIRE's development and held-out seed splits, using seeds 51–65 and 1–50 for LIBERO-PRO, and seeds 101–125 and 1–100 for Robosuite. Program generation, repair, and selection use the development split. Within each run, we freeze one accepted program per LIBERO-PRO task–setting pair and per Robosuite task. We then execute each program on every corresponding held-out seed using current observations and control APIs, without further program generation or repair. The execution archive and critic are used only during learning.

Implementation and compute. We deploy GLM-5.3-Flash locally on eight NVIDIA PRO 6000 GPUs, using the `max` reasoning setting and an agent sampling temperature of 1.0. Perception services include Molmo2 (Clark et al., 2026), SAM 3 (Carion et al., 2026), and GraspNet (Fang et al., 2020). These services and simulation workloads run on a separate pool of eight NVIDIA A100 GPUs. Learning updates the control programs and execution memory while keeping the language-model parameters fixed.

Simulation search and diagnostics. Each development configuration permits at most three main-search repairs and one additional follow-up repair. After each accepted source update, we sample at most $K = 3$ historical success witnesses for full-trace certification. We then run at most $k_{\mathrm{diag}} = 3$ diagnostic trials on uncertified configurations, as described in Section 3.3.

Comparison methods. We compare SCOPE with ASPIRE using the same GLM-5.3-Flash backbone and include ASPIRE's Opus 4.6 results (Lu et al., 2026). Other program-based baselines include CaP-Agent0 (Fu et al., 2026), RHO (Elmaaroufi et al., 2026), and RATs (Zhang et al., 2026). For LIBERO-PRO, the vision-language-action baselines include OpenVLA, π_0, and $\pi_{0.5}$ as reported by ASPIRE, together with X-VLA (Zheng et al., 2026), MolmoAct (Lee et al., 2026), and SURE-Flow (Islam et al., 2026). We use official LIBERO-PRO leaderboard results (LIBERO-PRO, 2026) for X-VLA and MolmoAct, and reported results for SUREFlow. We compute aggregate success for the comparison methods as the equal-weight mean of their reported suite–setting or task scores. The other comparison methods retain their original training and inference protocols. Table notes specify differences in evaluation trial counts and RHO's Robosuite seed split.

Independent runs and reporting. For our GLM-5.3-Flash simulation experiments, we repeat the complete heuristic-learning and held-out evaluation pipeline three times. Each repetition includes fresh program generation, development, and evaluation of the resulting frozen programs. The development and held-out seed sets remain fixed across runs. Success is determined by the benchmark task evaluator. Within each run, we average success over held-out seeds and then over tasks. LIBERO-PRO's overall score weights the six suite–setting combinations equally. We report means and sample standard deviations across the three run-level scores, capturing variation across complete learning runs. Simulation benchmark and ablation results are percentages with two decimal places. Results from the comparison methods' original reports are point estimates, and bold indicates the best value or mean before rounding, including ties.

C Ablation Protocol

The ablations in Table 4 use GLM-5.3-Flash on all ten tasks in each of Goal–Task and Spatial–Pos, the settings with the largest gains over the same-backbone baseline.

Controlled comparisons. The four conditions use the same development and held-out configurations, with three repetitions totaling 240 task-level learning runs. Each run independently generates its initial program and task graph and maintains its own archive and success bank. All use $K = 3$, $k_{\mathrm{diag}} = 3$, and the same per-configuration main-search and follow-up repair limits. Each ablation changes only the specified component. Acceptance and frozen-program evaluation follow the same protocol. ASPIRE is shown as a separate baseline.

Removing certification feedback. We mask full-trace certification results, first divergence details, and derived summaries from planning and code generation. Internal replay and the diagnostic and follow-up procedures remain unchanged, with observations and failures from environment diagnostics still available.

Removing failure-frontier guidance. We mask the diagnostic graph, node labels, failure frontiers, evidence windows, and derived summaries, while retaining raw observations, API traces, errors, outcomes, and source programs.

Removing explicit revision-level planning. We retain failure diagnosis and repair planning but omit the explicit choice among local repair, stage redesign, and strategy regeneration before generating a complete candidate program.

D Detailed Certification Rates and Costs

Table 5 provides the numerical results underlying Figure 4. The six LIBERO-PRO rows summarize ten tasks each, and the Robosuite row summarizes seven tasks. All quantities are task macro-averages. The time columns retain their respective denominators.

Table 5: **Certification rates and costs of SCOPE.** Task macro-averages (%) with GLM-5.3-Flash over 60 LIBERO-PRO task-setting pairs and seven Robosuite tasks.

Setting	Certification rate	Replay time / original execution time	Certification time / total learning time	Diagnostic trial time / total learning time
Object–Pos	60.6	4.93	0.54	0.87
Object–Task	55.3	5.53	0.83	1.41
Goal–Pos	49.0	6.12	0.75	1.28
Goal–Task	38.9	6.59	1.36	2.67
Spatial–Pos	42.1	6.25	1.04	2.04
Spatial–Task	41.9	5.94	0.98	2.12
Robosuite	44.7	4.62	1.05	3.10

Diagnostic trial time covers environment execution only. Total learning time includes model calls, environment executions, replay, and orchestration.

E Real-World Tasks and Protocol

We use the gripper-equipped Quanta X2 from X Square Robot, a wheeled humanoid platform with two 7-degree-of-freedom (DoF) arms, a 4-DoF waist, and a 2-DoF head (X Square Robot, n.d.b). The platform has a 2-DoF mobile base, stands 1.64 m tall, and provides a nominal arm reach of 765 mm (X Square Robot, n.d.a).

Figure 5 illustrates the three physical tasks evaluated in Table 3. The tasks cover base–arm coordination, bimanual coordination, and fine manipulation, respectively.

Both methods use ten development configurations per task and share the language model, perception modules, and robot control interfaces. ASPIRE (Lu et al., 2026) uses five search rounds with eight candidate programs per round. For each development configuration, SCOPE permits at most three main-search repairs and one additional follow-up repair after a failed physical diagnostic on a configuration left uncertified by replay (Section 3.3). After development, we freeze one program per method and task and evaluate it on 20 held-out initial placements without further repair.

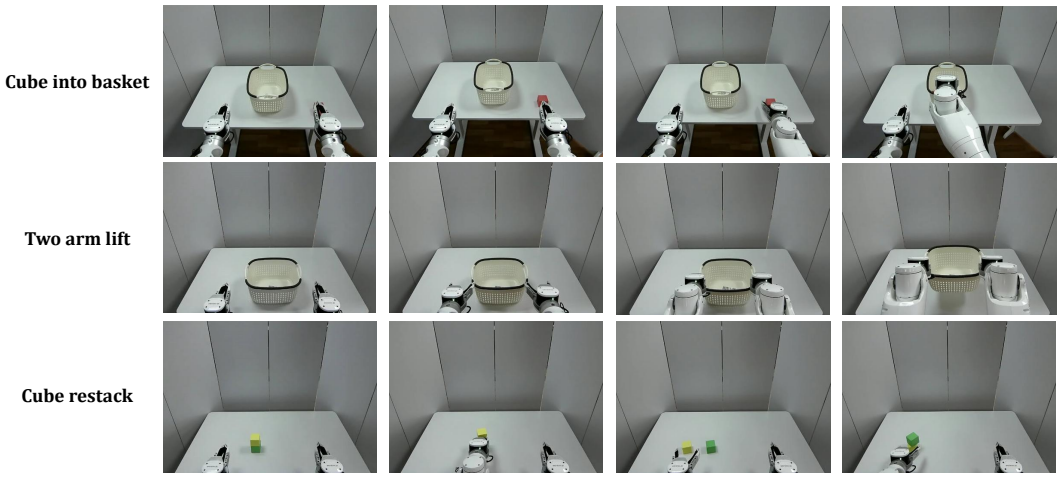


Figure 5: **Real-world task executions.** Representative frames progress from left to right for *Cube into basket* (top), *Two arm lift* (middle), and *Cube restack* (bottom). The tasks evaluate mobile manipulation with base–arm coordination, bimanual coordination, and fine manipulation, respectively.

F RESULTS BY LIBERO TASK

Tables 6 and 7 report individual task results for the Pos and Task settings, respectively, grouped by LIBERO-Object, LIBERO-Goal, and LIBERO-Spatial.

Table 6: Success rates (%) for individual LIBERO tasks under the Pos setting.

Task	ASPIRE Opus 4.6	ASPIRE GLM-5.3-Flash	**SCOPE** GLM-5.3-Flash
LIBERO-Object			
Alphabet soup	**100.00**	98.67 ± 1.15	**100.00** ± 0.00
BBQ sauce	**100.00**	99.33 ± 1.15	**100.00** ± 0.00
Butter	**100.00**	98.67 ± 2.31	99.33 ± 1.15
Chocolate pudding	**100.00**	98.67 ± 2.31	**100.00** ± 0.00
Cream cheese	**100.00**	98.00 ± 2.00	**100.00** ± 0.00
Ketchup	92.00	94.67 ± 4.16	**96.67** ± 3.06
Milk	92.00	96.67 ± 4.16	**98.67** ± 2.31
Orange juice	**100.00**	98.67 ± 1.15	**100.00** ± 0.00
Salad dressing	96.00	97.33 ± 1.15	**99.33** ± 1.15
Tomato sauce	98.00	**100.00** ± 0.00	**100.00** ± 0.00
LIBERO-Goal			
Open middle drawer	72.00	78.67 ± 13.61	**89.33** ± 4.16
Open top drawer + bowl	86.00	88.67 ± 3.06	**98.67** ± 1.15
Push plate to stove	0.00	2.00 ± 0.00	**3.33** ± 1.15
Bowl on plate	92.00	95.33 ± 3.06	**100.00** ± 0.00
Bowl on stove	96.00	98.67 ± 1.15	**100.00** ± 0.00
Bowl on cabinet top	**100.00**	99.33 ± 1.15	**100.00** ± 0.00
Cream cheese in bowl	74.00	79.33 ± 7.02	**92.67** ± 3.06
Wine on rack	98.00	98.67 ± 1.15	**100.00** ± 0.00
Wine on cabinet top	**100.00**	99.33 ± 1.15	**100.00** ± 0.00
Turn on stove	90.00	94.00 ± 4.00	**99.33** ± 1.15
LIBERO-Spatial			
Betw. plate + ramekin	94.00	96.67 ± 2.31	**99.33** ± 1.15
From table center	32.00	30.67 ± 4.62	**48.67** ± 13.01
In top drawer	10.00	20.00 ± 19.08	**28.67** ± 6.43
Next to cookie box	16.00	22.67 ± 2.31	**48.67** ± 6.43
Next to plate	**100.00**	98.00 ± 2.00	99.33 ± 1.15
Next to ramekin	20.00	13.33 ± 8.08	**21.33** ± 5.03
On cookie box	24.00	35.33 ± 9.45	**54.67** ± 2.31
On ramekin	94.00	94.00 ± 4.00	**98.00** ± 3.46
On stove	86.00	87.33 ± 11.02	**96.00** ± 3.46
On wooden cabinet	34.00	46.67 ± 10.26	**59.33** ± 13.32

Table 7: Success rates (%) for individual LIBERO tasks under the Task setting.

Task	ASPIRE Opus 4.6	ASPIRE GLM-5.3-Flash	**SCOPE** GLM-5.3-Flash
LIBERO-Object			
Alphabet soup	**100.00**	98.00 ± 2.00	**100.00** ± 0.00
BBQ sauce	**100.00**	99.33 ± 1.15	**100.00** ± 0.00
Butter	98.00	98.67 ± 2.31	**100.00** ± 0.00
Chocolate pudding	**100.00**	98.67 ± 1.15	99.33 ± 1.15
Cream cheese	72.00	84.00 ± 5.29	**91.33** ± 4.16
Ketchup	**96.00**	88.00 ± 19.08	94.00 ± 0.00
Milk	**100.00**	98.67 ± 1.15	98.67 ± 1.15
Orange juice	**100.00**	99.33 ± 1.15	99.33 ± 1.15
Salad dressing	92.00	94.00 ± 5.29	**96.67** ± 1.15
Tomato sauce	96.00	97.33 ± 3.06	**98.00** ± 2.00
LIBERO-Goal			
Open middle drawer	0.00	9.33 ± 11.02	**30.00** ± 10.00
Open top drawer + bowl	46.00	59.33 ± 9.24	**80.00** ± 6.93
Push plate to stove	76.00	67.33 ± 20.53	**81.33** ± 13.32
Bowl on plate	88.00	84.67 ± 12.06	**90.67** ± 9.45
Bowl on stove	0.00	8.67 ± 8.33	**22.00** ± 9.17
Bowl on cabinet top	4.00	7.33 ± 5.77	**14.00** ± 12.17
Cream cheese in bowl	74.00	72.67 ± 22.48	**89.33** ± 9.87
Wine on rack	86.00	94.00 ± 4.00	**100.00** ± 0.00
Wine on cabinet top	14.00	23.33 ± 15.53	**36.00** ± 17.32
Turn on stove	66.00	61.33 ± 28.10	**75.33** ± 5.77
LIBERO-Spatial			
Betw. plate + ramekin	8.00	12.00 ± 8.00	**18.00** ± 4.00
From table center	64.00	66.67 ± 13.61	**80.00** ± 9.17
In top drawer	**96.00**	92.00 ± 5.29	90.67 ± 10.07
Next to cookie box	94.00	96.67 ± 1.15	**100.00** ± 0.00
Next to plate	70.00	77.33 ± 9.45	**82.00** ± 8.72
Next to ramekin	60.00	**64.00** ± 24.25	62.67 ± 15.01
On cookie box	70.00	71.33 ± 10.26	**92.00** ± 5.29
On ramekin	46.00	61.33 ± 13.32	**74.00** ± 9.17
On stove	42.00	51.33 ± 7.02	**58.00** ± 6.00
On wooden cabinet	46.00	52.67 ± 11.02	**54.67** ± 6.11

G BEHAVIORAL INTERFERENCE OF PROGRAM EDITS

Fix programs P, P' and a task distribution $\mathcal{D}_d$. Compare their executions on the same instance $\xi = (x_0, \omega, \kappa)$, sharing initial conditions, randomness, and execution semantics. Under the assumptions in Section 2, identical external interactions, including termination, yield the same binary task outcome. Let $E_{P,P'}$ denote the event that these interactions differ, and define the edit's behavioral divergence probability as $\eta(P, P') = \Pr(E_{P,P'})$. All probabilities and expectations below are over $\xi \sim \mathcal{D}_d$.

Proposition (Behavioral divergence bounds). The change in expected success and the probability of losing a prior success satisfy

$$\begin{aligned} J(P') - J(P) &= \mathbb{E}\big[(Y(P';\xi) - Y(P;\xi))\mathbf{1}[E_{P,P'}]\big], \\ |J(P') - J(P)| &\le \eta(P, P'), \\ \Pr\big(Y(P;\xi) = 1,\, Y(P';\xi) = 0\big) &\le \eta(P, P'). \end{aligned} \tag{7}$$

The last probability measures regressions across all task instances.

Proof. Outside $E_{P,P'}$, the outcomes agree, so multiplying their difference by $\mathbf{1}[E_{P,P'}]$ leaves it unchanged. Taking expectations gives the first identity. Since binary outcomes differ by at most one,

$$|J(P') - J(P)| \le \mathbb{E}\big[|Y(P';\xi) - Y(P;\xi)|\mathbf{1}[E_{P,P'}]\big] \le \Pr(E_{P,P'}).$$

The regression event $\{Y(P;\xi) = 1,\, Y(P';\xi) = 0\}$ is a subset of $E_{P,P'}$, giving the final bound.

Edits within a conditional branch. Suppose P and P' differ only within a branch guarded by the same side-effect-free predicate g. Let $q_P(g)$ be the probability that P reaches this guard and enters the branch at least once during execution. The programs execute identically before the first entry. If the branch is never entered, the modified code is never executed. Thus,

$$E_{P,P'} \subseteq \{P \text{ enters the guarded branch at least once}\}, \qquad \eta(P, P') \le q_P(g).$$

Implications for repair. An improvement $J(P') - J(P) = \Delta > 0$ requires $\eta(P, P') \ge \Delta$. A low behavioral divergence probability therefore limits the possible gain, while the outcomes of changed executions determine whether performance improves. This motivates broader revisions when local repairs are insufficient. SCOPE selects revision levels from execution evidence and uses replay to assess preservation on recorded instances.

H CERTIFICATION GUARANTEE AND FEEDBACK COST

H.1 INSTANCE-SPECIFIC CERTIFICATION

Fix a historical success witness $w = (P_w, \xi_w, T_w)$ with $Y(P_w; \xi_w) = 1$, and let P be the program being checked. The certificate $C_t(P, w) = 1$ requires a complete match against the stateful tape T_w and normal termination, as defined in Section 3.3. Under the replay assumptions in Section 2,

$$C_t(P, w) = 1 \quad \Longrightarrow \quad Y(P; \xi_w) = 1. \tag{8}$$

Proof. Compare replay of the tested program P against T_w with its execution in the environment on the original instance ξ_w. They begin with the same program inputs and runtime semantics. We induct over API events, maintaining agreement between their program-visible values and reference relationships, and between the real execution's external history and the witness tape.

Before the first event, the inputs agree and the external history is empty. Suppose the correspondence holds before event j. Deterministic program execution produces the same API request and pre-call public state in both runs. Since replay matches, these equal $(q_{w,j}, B^{-}_{w,j})$. The real environment starts from the same instance and has received the same preceding interactions as the witness. Its response and public-state effects therefore match the recorded $(v_{w,j}, B^{+}_{w,j})$. Faithful replay reproduces the return value or exception and the associated mutations, preserving shared references and array views. The two runs of P consequently agree after the event, establishing the induction step.

A full match consumes every recorded event without additional external interactions. Deterministic continuation then makes the real execution terminate normally, as replay does. The resulting environment state matches that of the successful witness, so the fixed evaluator gives $Y(P; \xi_w) = Y(P_w; \xi_w) = 1$.

The certificate applies to the recorded instance ξ_w. A fresh reset of the same configuration may have different inputs or randomness. A divergent or unavailable replay does not establish task failure. Source acceptance remains determined by success on the current configuration. Certification provides evidence for later repairs, without requiring monotonic improvement in $J(P_t^{\mathrm{acc}})$ or preservation of every historical success.

H.2 FEEDBACK COST

Let n_t^{tr} and n_t^{diag} count selected replay checks and diagnostic environment executions after search trial t. The T search trials include follow-up executions. Diagnostics are counted separately. The per-update limits imply $n_t^{\mathrm{tr}} \le K A_t$ and $n_t^{\mathrm{diag}} \le k_{\mathrm{diag}} A_t$, so both counts are zero when the accepted source is unchanged. These limits give

$$\begin{aligned} Q_{\mathrm{tr}} &= \sum_{t=1}^{T} n_t^{\mathrm{tr}} \le K \sum_{t=1}^{T} A_t, \\ Q_{\mathrm{diag}} &= \sum_{t=1}^{T} n_t^{\mathrm{diag}} \le k_{\mathrm{diag}} \sum_{t=1}^{T} A_t. \end{aligned} \tag{9}$$

Let c_{tr} and c_{env} denote the mean observed costs of a selected replay check and a diagnostic environment execution over the run. Their aggregate cost is

$$C^{\mathrm{fb}} = Q_{\mathrm{tr}} c_{\mathrm{tr}} + Q_{\mathrm{diag}} c_{\mathrm{env}}. \tag{10}$$

The count bounds are independent of the success-bank size, while replay time depends on trace length, the first divergence, and local program execution. Total learning cost additionally includes search executions, model queries, record processing, and orchestration. Follow-up executions are included among the search trials. Their diagnosis, planning, and generation requests contribute to model-query cost. The certification timing reported in Section 4.4 also includes retrieval, record verification, and self-matching in addition to the selected replay checks.

I AI USE STATEMENT

We used large language models to assist with language editing and polishing of the manuscript. The authors take full responsibility for its content.