

Trace-to-Architecture: Capability Bottleneck Diagnosis for Harness Revision

Wenfeng Wang^{1*} Jianrui Shi^{2*†} Mingang Guo¹ Jiayuan Ma²
Yuqi Lu² Jiajun Sun² Guojiang Zhao³ Jing Li²

¹University of Pennsylvania, Philadelphia, USA

²Harbin Institute of Technology, Shenzhen, China

³DP Technology, Beijing, China

fw93810@seas.upenn.edu jianrui.shi@stu.hit.edu.cn

Abstract

Improving an LLM agent requires knowing what to change when it fails. Similar failures can require different fixes, and resolving one problem can expose another. We introduce Trace-to-Architecture (T2A), a framework that uses execution trajectories to determine what to change in an agent’s harness and when those changes apply. A development agent diagnoses failures using execution evidence from successful and failed tasks, revises the corresponding harness components, and then checks whether target tasks improve, selected successful tasks remain successful, and the edits are actually triggered. New execution evidence guides subsequent diagnosis and revision. Across ALFWorld, ScienceWorld, and AppWorld, T2A achieves the highest aggregate score for all three evaluated task agents, exceeding their respective strongest baselines by 6.2–30.3 points. We also integrate T2A’s diagnosis and validation procedures into Meta-Harness, improving the resulting harnesses’ mean ScienceWorld score by 21.6 points from the same starting harness.

1 Introduction

Recursive self-improvement (RSI) aims to let agents turn accumulated experience into lasting improvements in their capabilities and in how they pursue further improvement (Zelikman et al., 2024; Zhang et al., 2026b; Duan et al., 2026; Zheng et al., 2026). For agents built around a large language model (LLM), one practical route is to revise the agent harness: components such as prompts, context management, stored memory, and tool interfaces. These components govern how

the model interacts with its environment. Revising the harness can improve an agent’s task performance without updating model weights. MetaRSI incorporates such revisions alongside changes to data and model parameters (Tan et al., 2026), while other systems automate harness revision using execution feedback (Lee et al., 2026; Zhang et al., 2026a; Xia et al., 2026). SoL-Pi further demonstrates how improvements to execution and context handling can reduce inference cost (Liu et al., 2026).

Optimizing a harness requires choosing among these options based on the current execution problems. Task outcomes alone provide limited guidance for this decision. Similar failures may arise from causes such as missing knowledge, failed delivery of memory, or inadequate execution rules, each requiring a different revision. Fixing one problem can also expose another or disrupt previously successful behavior. Records of model inputs and outputs, actions, and environment feedback help distinguish these causes (Barke et al., 2026; Chen et al., 2026b). Effective harness revision therefore requires checking diagnoses across related tasks, validating the effects of edits, and using new execution evidence to identify the next problem to address.

To this end, we introduce **Trace-to-Architecture (T2A)**, a framework for identifying what to change in a harness and checking whether those changes address the diagnosed failures. Within T2A, a task agent runs tasks under the current harness. A development agent compares related successful and failed execution trajectories to identify failure causes and determine which harness components to change and when those changes apply. Before evaluation, the agent

*Equal contribution.

†Corresponding author.

records acceptance conditions covering target-task improvement, preservation of selected successful tasks, and execution evidence that the edits were triggered. All three are required for acceptance. Changes that pass validation are retained, and new execution trajectories help identify the next problems to address.

We evaluate T2A with three task agents in ALFWorld for household tasks, ScienceWorld for scientific experiments, and AppWorld for cross-application tasks (Shridhar et al., 2021; Wang et al., 2022; Trivedi et al., 2024). T2A achieves the highest aggregate score across these environments for each task agent among the evaluated methods. With the starting harness and Meta-Harness’s outer evaluation and selection procedure fixed, integrating T2A’s diagnosis and validation requirements raises the resulting harnesses’ mean ScienceWorld score by 21.6 points.

2 Related Work

Agent and harness improvement. Self-improving agents revise programs, experience, or optimization strategies (Zelikman et al., 2024; Zhang et al., 2026b; Tan et al., 2026). With model weights fixed, GEPA adapts prompts using execution feedback (Agrawal et al., 2026), while Reflexion, CLIN, and WorldEvolver update experience or memory (Shinn et al., 2023; Majumder et al., 2023; Zhang et al., 2026c). ADAS and AFlow search over agent programs and workflows (Hu et al., 2025; Zhang et al., 2025). Meta-Harness extends this search to harness code, using historical implementations, scores, and trajectories to propose revisions (Lee et al., 2026).

Diagnosis and evaluation. AgentRx, TraceElephant, and AgentDebugX study how execution records can support failure localization, attribution, and recovery (Barke et al., 2026; Chen et al., 2026b; Zhu et al., 2026). Harness evaluation also distinguishes producing updates from benefiting from them (Lin et al., 2026b) and tests generalization under matched development budgets (Wang et al., 2026). Accordingly, we evaluate frozen-system task outcomes, changes in target errors, and development protocols under controlled conditions.

Harness revision. Prior work uses failure diagnosis to guide system modifications. Self-Harness summarizes recurring causes of failure and re-

stricts changes to relevant components (Zhang et al., 2026a). Agentic Harness Engineering and HarnessX record the reasons and expected effects of changes and check those changes against execution evidence and task outcomes (Lin et al., 2026a; Chen et al., 2026c). HarnessFix locates the steps and system components responsible for errors and checks whether repairs reduce the targeted errors (Chen et al., 2026a). T2A likewise revises harnesses based on execution evidence, emphasizing the diagnosis of capability bottlenecks across tasks and the conditions under which they arise to guide revisions and formulate validation conditions. Appendix A compares the workflows in detail.

3 Method

3.1 Overview and Roles

T2A alternates between *data collection* (Section 3.2) and *harness revision* (Section 3.3); Figure 1 shows the process, and Algorithm 1 in Appendix B summarizes the complete loop. In each round, the task agent runs tasks under the current harness and produces execution trajectories. The development agent then analyzes these trajectories and revises the harness. Development is bounded by the stopping conditions and resource limits specified for each experiment. Appendices D.1 and D.2 describe the starting-harness requirements and task splits.

3.2 Data Collection: Execution and Trajectory Collection

In round r , tasks run under a fixed harness

$$H_r = (D_r, \mathcal{C}_r, R_r). \quad (1)$$

The driver D_r executes components, constructs model inputs from the current task state, and processes proposed actions before execution. Its editable scope includes input construction and action processing, but excludes the scorer and externally enforced resource limits. $\mathcal{C}_r$ contains editable components and their invocation conditions, such as skill cards, within-episode memory, and action checks. Skill cards are texts that store task knowledge or operational experience. When their invocation conditions are met, relevant cards are added to the model input. R_r specifies what information is recorded during execution and when it is recorded.

For task d , the interaction records $e_{d,t}$ collected according to R_r , together with the terminal score

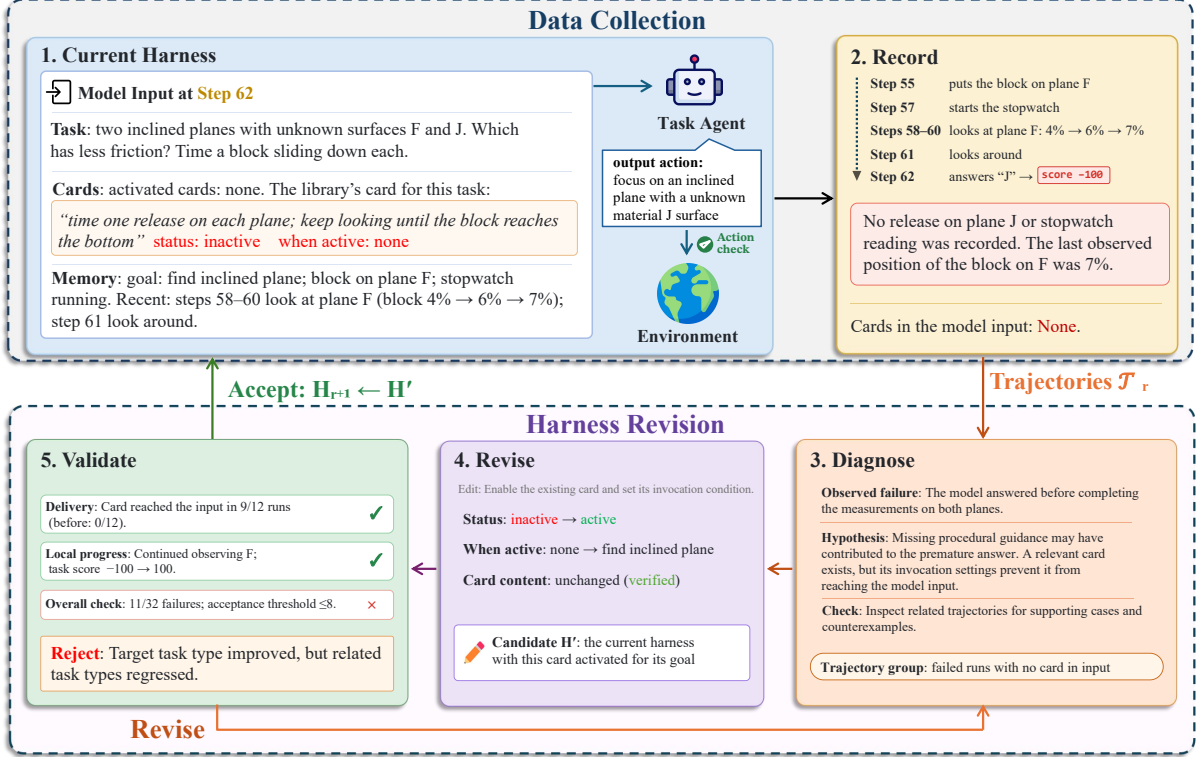


Figure 1: T2A alternates between trajectory collection (top) and diagnosis, revision, and validation (bottom). In the example, execution evidence identifies a skill card that is absent from the model input. Adjusting its invocation conditions improves the target task, but regressions on related tasks lead to rejection and further revision.

s_d , form a trajectory:

$$\begin{aligned} \tau_d &= ((e_{d,1}, \dots, e_{d,n_d}), s_d), \\ \mathcal{T}_r &= \{\tau_d : d \in \mathcal{Q}_r\}. \end{aligned} \quad (2)$$

τ_d denotes the execution trajectory for task d , and $\mathcal{T}_r$ is the set of trajectories collected in round r . $\mathcal{Q}_r$ is the development task set used in that round, and n_d is the number of interactions for task d .

3.3 Harness Revision

Diagnosis. In each round, T2A selects all tasks that did not receive full scores, giving $\mathcal{F}_r = \{\tau_d \in \mathcal{T}_r : s_d < s_d^{\max}\}$, where $s_d^{\max}$ is the full score for task d . Scripts group these trajectories into $\mathcal{G}_r$ using observable execution features, retaining task identifiers and execution steps for context retrieval. The development agent examines representative trajectories from each group, using model inputs, decisions, action processing, and environment feedback to identify possible causes of failure and the situations in which they apply. These form the candidate diagnosis set $\mathcal{Z}_r^{\text{cand}}$. Each diagnosis record takes the form

$$z = (h_z, c_z, \mathcal{E}_z). \quad (3)$$

Here h_z is a hypothesis about the cause of failure, describing how the observed problem relates to the information available at that point or to harness operations; c_z specifies its applicability conditions; and $\mathcal{E}_z$ contains the concrete execution records used to check the hypothesis across related successful and failed tasks.

The agent searches the execution trajectory set $\mathcal{T}_r$ for related cases and checks each hypothesis in its execution context. The agent revises or withdraws explanations according to the evidence; problems whose causes cannot be determined from the available records remain unresolved. Diagnoses supported by execution records are consolidated for the same problem and kept separate for distinct problems, forming $\mathcal{Z}_r$. Every task that lost points is assigned to a diagnosis cluster or marked as unresolved; unresolved problems retain open questions and the additional records needed. The resulting records link diagnoses to relevant tasks and steps to guide revision. For example, in Figure 1, the required guidance is stored in a skill card, but the card is disabled for the current invocation and absent from the model input, identifying invocation settings as the revision tar-

get.

Revision. The development agent translates supported diagnoses $\mathcal{Z}_r$ into a harness edit δ , linking the edit to the affected tasks and execution steps and describing its expected effects. Applying δ to the current harness H_r produces a candidate harness H' by modifying the driver, components, or logging rules. Component edits may modify existing components or introduce new ones. The recorded links between diagnoses, edits, and expected effects guide subsequent validation.

Revision targets depend on where execution breaks down. Possible edits include:

- **Skill-card content:** Revise the card when knowledge is missing or its procedure is unsuitable.
- **Guidance delivery:** Adjust retrieval or invocation conditions when applicable guidance is absent from the model input.
- **Decision rules:** Clarify operating rules when decision errors recur.
- **Action check:** Add checks based on observable prerequisites for invalid actions.
- **Memory and execution control:** Adjust memory updates or execution control when observations are outdated or known states are repeatedly visited.

When the available records leave the cause unresolved, the agent revises R_r to collect additional execution records or runs a targeted comparison before choosing a corrective change. Appendix E.3 illustrates how diagnosis and validation shift a revision from skill-card content to execution control.

Validation. Before running validation, the development agent records acceptance conditions for comparing the candidate harness H' with the current harness H_r . These specify the target tasks and their expected results, protected tasks that currently succeed and may be affected by the edit, and how to verify that the edit was actually triggered. Checks of specific target behaviors are included when specified by the validation plan.

Using execution records from both harnesses, the development agent judges the candidate against these conditions:

- **Target-task improvement:** Whether the target tasks improve as specified in the acceptance conditions.

- **Regression protection:** Whether the protected tasks retain their successful outcomes.
- **Edit triggering:** Whether execution records confirm that the edit was actually triggered, through component invocation, model inputs, or the relevant code path.

A candidate that fails validation may be retained for further revision or validation without being accepted. Acceptance requires satisfying all conditions of the applicable plan, registered before that validation run. Until acceptance, H_r remains the accepted harness. The development agent writes the acceptance conditions and judges whether they are met from execution records, without researcher review; the program does not independently adjudicate semantic acceptance. The agent saves the validation plan, execution records, and verdict whether or not the candidate is accepted. After acceptance, trajectories produced by the new harness are used to begin the next round of diagnosis.

4 Experiments

We evaluate T2A with Qwen3.5-27B, Qwen3.5-9B (Qwen Team, 2026), and DeepSeek V4.1 Flash (Xu et al., 2026) on three interactive environments: ALFWorld (Shridhar et al., 2021) for household tasks, ScienceWorld (Wang et al., 2022) for scientific experiments, and AppWorld (Trivedi et al., 2024) for cross-application tasks.

4.1 Experimental Setup

Environments and metrics. ALFWorld (Shridhar et al., 2021) requires agents to navigate rooms, find objects, and manipulate them to satisfy household goals, such as heating an object and placing it in a specified receptacle. ScienceWorld (Wang et al., 2022) requires agents to locate materials, operate equipment, and use experimental observations to complete scientific tasks and select answers. AppWorld (Trivedi et al., 2024) requires agents to write Python code to query and modify data across applications according to user requests. ALFWorld and AppWorld report task completion rates; ScienceWorld reports mean terminal score and fatal-failure rate.

Baselines. Minimal prompting supplies the task and interaction conventions. All methods in Table 1 use the minimal-prompt harness as their starting point. ReAct (Yao et al., 2023) uses a

Benchmark	Baselines						Ours	Δ
	Minimal prompt	ReAct	SkillNet	GEPA	WorldEvolver	Meta-Harness	T2A	
Qwen3.5-27B								
ScienceWorld	23.1 $\pm$ 1.6	11.0 $\pm$ 1.1	31.1 $\pm$ 1.3	58.6 $\pm$ 0.8	29.5 $\pm$ 1.8	63.7 $\pm$ 0.5	87.7 $\pm$ 0.1	+24.0
AppWorld Normal	58.9 $\pm$ 2.1	33.3 $\pm$ 2.7	–	86.1 $\pm$ 1.7	57.1 $\pm$ 1.8	79.8 $\pm$ 1.2	90.1 $\pm$ 1.2	+4.0
AppWorld Challenge	51.9 $\pm$ 0.5	18.9 $\pm$ 1.1	–	72.2 $\pm$ 1.6	47.5 $\pm$ 0.7	65.9 $\pm$ 0.7	74.0 $\pm$ 1.5	+1.8
ALFWorld Seen	65.2 $\pm$ 1.8	67.9 $\pm$ 1.4	89.3 $\pm$ 0.7	90.0 $\pm$ 0.7	64.3 $\pm$ 1.4	91.7 $\pm$ 0.8	97.4 $\pm$ 0.4	+5.7
ALFWorld Unseen	69.7 $\pm$ 2.6	67.2 $\pm$ 2.2	88.1 $\pm$ 0.7	94.8 $\pm$ 0.7	67.9 $\pm$ 2.2	93.5 $\pm$ 0.4	99.3 $\pm$ 0.7	+4.5
Average	48.65	34.88	–	76.72	49.30	76.38	89.37	+12.65
DeepSeek V4.1 Flash								
ScienceWorld	36.2 $\pm$ 2.0	54.8 $\pm$ 1.4	58.1 $\pm$ 1.4	64.8 $\pm$ 0.9	41.3 $\pm$ 1.7	66.6 $\pm$ 0.7	85.3 $\pm$ 0.6	+18.7
AppWorld Normal	82.7 $\pm$ 1.6	89.9 $\pm$ 1.2	–	88.7 $\pm$ 1.0	72.0 $\pm$ 1.8	91.5 $\pm$ 0.9	92.5 $\pm$ 1.2	+1.0
AppWorld Challenge	78.8 $\pm$ 0.9	84.4 $\pm$ 1.4	–	86.3 $\pm$ 0.7	76.5 $\pm$ 0.9	83.8 $\pm$ 0.3	84.3 $\pm$ 0.3	–2.0
ALFWorld Seen	78.6 $\pm$ 2.6	77.9 $\pm$ 1.4	97.9 $\pm$ 0.0	87.9 $\pm$ 0.7	61.4 $\pm$ 2.5	99.8 $\pm$ 0.4	98.1 $\pm$ 0.4	–1.7
ALFWorld Unseen	77.4 $\pm$ 1.6	76.9 $\pm$ 2.0	97.0 $\pm$ 0.0	90.3 $\pm$ 0.7	61.9 $\pm$ 2.0	99.0 $\pm$ 0.4	99.0 $\pm$ 0.4	0.0
Average	64.98	73.12	–	80.47	59.07	84.55	90.75	+6.20
Qwen3.5-9B								
ScienceWorld	–2.6 $\pm$ 3.1	–51.6 $\pm$ 4.9	–16.9 $\pm$ 2.0	22.4 $\pm$ 1.1	6.7 $\pm$ 2.9	39.3 $\pm$ 1.1	58.3 $\pm$ 0.9	+19.0
AppWorld Normal	4.2 $\pm$ 1.2	18.5 $\pm$ 1.0	–	33.3 $\pm$ 1.0	5.4 $\pm$ 1.0	11.3 $\pm$ 1.0	69.4 $\pm$ 1.5	+36.1
AppWorld Challenge	3.7 $\pm$ 0.4	9.4 $\pm$ 0.7	–	22.8 $\pm$ 0.7	2.9 $\pm$ 0.6	7.7 $\pm$ 0.6	50.3 $\pm$ 1.0	+27.5
ALFWorld Seen	20.5 $\pm$ 2.7	25.7 $\pm$ 1.9	62.9 $\pm$ 1.2	62.9 $\pm$ 1.4	17.1 $\pm$ 2.6	66.7 $\pm$ 1.1	89.0 $\pm$ 1.1	+22.3
ALFWorld Unseen	23.6 $\pm$ 3.0	18.7 $\pm$ 2.0	67.9 $\pm$ 0.7	69.4 $\pm$ 1.3	16.4 $\pm$ 2.2	68.9 $\pm$ 1.1	89.8 $\pm$ 0.9	+20.4
Average	7.80	-5.15	–	38.87	9.20	38.87	69.18	+30.32

Table 1: Test results (mean $\pm$ sample standard deviation over three evaluations). Bold marks the highest mean in each row. Average weights the three environments equally, averaging the two splits within AppWorld and ALFWorld. Δ is T2A’s difference from the highest baseline mean in the same row and model block. SkillNet has no AppWorld results or Average because its public release provides no AppWorld skill library or evaluation setup.

fixed reasoning–action–observation loop. WorldEvolver (Zhang et al., 2026c), SkillNet (Liang et al., 2026), GEPA (Agrawal et al., 2026), and Meta-Harness (Lee et al., 2026) represent memory, skill, prompt, and harness approaches. WorldEvolver is adapted to build memory on development tasks, freeze it, and execute each test task once. Baseline adaptations are in Appendix D.1.

Implementation details. We use Qwen3.5-27B and Qwen3.5-9B (Qwen Team, 2026), and DeepSeek V4.1 Flash (Xu et al., 2026) in all three environments to cover different levels of baseline task performance and examine how the initial distribution of capability bottlenecks affects subsequent revision outcomes. The development agent uses Claude Opus 5 (Anthropic, 2026b) in Claude Code (Anthropic, 2026a) across all three environments. T2A initialization augments an existing harness with structured execution records needed for diagnosis and gives the development agent access to modify relevant components and their in-

vo- cation points. Detailed initialization requirements are described in Appendix D.1.

For each task agent and environment, every method in Table 1 is evaluated three times using the same frozen harness. The table reports the means and sample standard deviations of these three evaluations.

4.2 Overall Performance

Table 1 compares the resulting frozen systems.

T2A’s gains are especially pronounced for task agents and environments with weaker baseline performance. With the three environments weighted equally, T2A achieves the highest aggregate score for all three task agents, exceeding the best baseline for Qwen3.5-27B, DeepSeek V4.1 Flash, and Qwen3.5-9B by 12.65, 6.20, and 30.32 points, respectively (Table 1, Δ). The initially weakest task agent, Qwen3.5-9B, obtains the largest aggregate gain. For Qwen3.5-27B and DeepSeek V4.1 Flash, the main gains come from

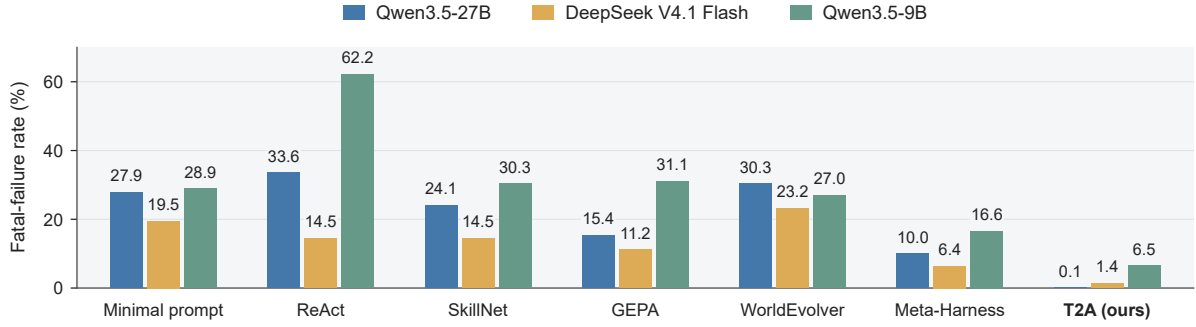


Figure 2: ScienceWorld fatal-failure rates, averaged over three evaluations. Lower is better; colors distinguish the three task models.

ScienceWorld, where baseline performance is relatively weak: mean gains over minimal prompting are 64.6 and 49.1 points, respectively.

The same intervention has different effects across task agents and environments. ReAct (Yao et al., 2023) improves DeepSeek V4.1 Flash’s ScienceWorld mean but lowers both Qwen task agents’ means and increases their fatal-failure rates. WorldEvolver (Zhang et al., 2026c) improves ScienceWorld means for all three task agents but falls below minimal prompting in ALF-World for each. These differences show that the same intervention does not guarantee consistent improvement. Section 4.3.1 uses failure distributions and records of diagnosis, revision, and validation to examine problems across settings and successive harness versions, together with the corresponding revision targets.

A higher mean score does not imply that all current capability bottlenecks have been alleviated. WorldEvolver improves the ScienceWorld means of Qwen3.5-27B and DeepSeek V4.1 Flash while increasing their fatal-failure rates. For Qwen3.5-27B, this rate rises from 27.9% to 30.3%; T2A instead reduces it to 0.1% while improving the mean (Figure 2). However, these aggregate metrics cannot determine whether a revision alleviates its targeted problem. Section 4.3.2 uses component ablations to examine changes in scores and target errors; Appendices C and E.2 provide score decompositions and cases where later failures obscure local progress, respectively.

4.3 Analysis

4.3.1 Identifying and Addressing Capability Bottlenecks

We evaluate the minimal-prompt starting harness, an intermediate harness version, and the final T2A harness once each on the same test tasks within each environment, and classify failures from their execution trajectories. The intermediate version is the accepted harness snapshot nearest the middle of each harness revision sequence. Figure 3 shows failure distributions across task agents and environments and how they change across harness versions. Classification and counting rules are in Appendix D.5.

Different failure patterns call for different revision targets. Under minimal prompting, Qwen3.5-9B frequently encounters action-syntax errors in ScienceWorld and output-protocol or authentication failures in AppWorld. DeepSeek V4.1 Flash has fewer such failures, with ScienceWorld losses concentrated more in task procedure and answer errors. For Qwen3.5-9B in ALF-World, finding the target object is a major obstacle. The records show revisions to output-format and action-syntax handling for Qwen3.5-9B in ScienceWorld, refined planting and life-stage experience for Qwen3.5-27B, and action checks for DeepSeek V4.1 Flash’s incorrect focus actions.

Dominant failure types change during revision, prompting renewed diagnosis of capability bottlenecks. Although overall task scores improve across the selected snapshots, error categories do not change in parallel. In ScienceWorld, Qwen3.5-9B’s action-syntax errors decline across the three snapshots, but procedural errors remain prominent in the intermediate version and decrease only in

		ScienceWorld							AppWorld										ALFWorld					
		Format	Action syntax	Loop / memory	Procedure	Fact / answer	Fatal focus	Missed focus	Output protocol	Auth	API usage	Execution	No action	Missing items	Extra items	Side effects	Wrong edit	Wrong answer	No interaction	Not found	Seen, not taken	Not processed	Not placed	One of two
Initial	DeepSeek	8	6	7	37	18	18	29	1	1	0	2	0	3	4	2	4	2	0	8	3	7	0	2
	Qwen-27B	0	21	8	42	26	28	21	3	1	1	4	1	9	9	3	12	3	4	3	2	22	0	<1
	Qwen-9B	13	44	21	56	26	29	51	25	18	6	9	7	15	5	2	5	4	2	41	9	16	7	3
Intermediate	DeepSeek	1	5	<1	35	13	9	17	2	0	<1	4	<1	2	3	3	1	0	0	4	0	2	<1	<1
	Qwen-27B	0	18	12	29	18	9	17	0	6	4	7	0	9	6	<1	11	5	3	0	0	17	0	3
	Qwen-9B	15	27	27	61	23	21	39	17	7	4	4	5	17	6	0	6	0	0	20	10	7	5	0
Final	DeepSeek	1	2	9	22	2	2	7	1	<1	0	1	0	2	3	2	3	1	0	<1	0	0	0	0
	Qwen-27B	<1	9	1	21	0	0	11	1	<1	0	1	0	3	7	2	6	<1	<1	<1	0	1	0	0
	Qwen-9B	0	17	13	41	7	7	32	<1	<1	2	1	1	9	12	2	16	2	0	6	1	2	<1	<1

Figure 3: Failure categories for initial (minimal-prompt), intermediate, and final T2A harnesses, from one evaluation per harness. Cells show percentages of all test tasks in each environment; < 1 denotes less than 1%. Definitions and counting rules are in Appendix D.5.

the final version. The Qwen3.5-27B task agent shows a similar pattern: fatal focus errors decline, whereas loop and memory errors increase in the intermediate version before falling. Paired task analysis in AppWorld shows that, of 384 tasks where Qwen3.5-9B was initially blocked by problems in generating or executing application calls, 216 succeed under the final harness, 150 shift to content-level failures, and only 18 remain in the original categories. As earlier obstacles diminish, problems in object selection and modification content become more prominent. Aggregate task scores alone therefore cannot establish whether a revision alleviates its targeted problem or identify other bottlenecks exposed as earlier obstacles diminish. Appendix E.1 provides a sequence of harness revisions from ALFWorld–DeepSeek V4.1 Flash, showing how T2A uses failures remaining after revision to diagnose bottlenecks anew and adjust subsequent revisions and validation.

4.3.2 Effects of Targeted Revisions on Capability Bottlenecks

In the ScienceWorld–DeepSeek V4.1 Flash system reported in Table 1, skill cards, action checks, and within-episode memory address fact/answer errors, erroneous focus actions, and repeated observations, respectively. We remove these components individually and in combination, retaining the other components and basic execution functions, and compare task scores on the same 241

Removed	Mean	Δ
None (full)	85.2	—
Cards	65.0	−20.2
Check	70.4	−14.8
Memory	70.9	−14.3
Cards + check	52.3	−32.9
Cards + memory	46.5	−38.7
Check + memory	57.4	−27.8
All components	37.7	−47.5

Table 2: Individual and combined component ablations on ScienceWorld. Δ is the task-paired score difference from the full system. Evaluation and implementation details are in Appendix D.3.

test tasks. Individual-component ablations also examine changes in the corresponding target errors (Appendix D.3, Table 5). Table 2 reports the complete score results; evaluation settings and target-error definitions are in Appendix D.3.

The components selected through diagnosis alleviate their targeted capability bottlenecks. Removing skill cards, action checks, and within-episode memory raises the respective counts of tasks with fact/answer errors, focus actions that the rules would block, and repeated observations from 4, 0, and 14 to 28, 30, and 48 (Appendix D.3, Table 5). Combined removals cause larger performance losses than the corresponding individual removals, indicating that final performance depends

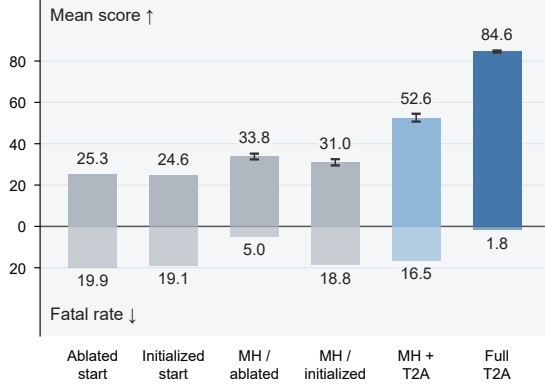


Figure 4: ScienceWorld development comparisons with DeepSeek V4.1 Flash. Mean scores appear above zero and fatal-failure rates (%) below. Results are averaged over three independent development runs; error bars show sample standard deviations of scores. Starting harnesses are evaluated once. MH: Meta-Harness.

on contributions from multiple revisions (Table 2). Each component contributes alongside the other revisions; retaining any single component does not preserve the full system’s performance.

Further trajectory checks show that all newly fatal tasks after card or action-check removal contain the corresponding target error, whereas this is not the case after memory removal. Trajectories are therefore needed to examine the link between target errors and final score losses; a score decrease alone does not identify its specific cause (Appendix D.3).

4.3.3 Starting Harnesses, Instructions, and Workflow

We conduct ablation experiments on ScienceWorld using DeepSeek V4.1 Flash as the task agent. Under a shared development budget, we use Meta-Harness as the base development workflow to examine the effects of individual design choices in T2A. We call the harness augmented to meet T2A’s execution-record requirements the initialized start and construct an ablated start from it. To examine the effects of support functions provided at initialization, we remove the record-keeping and execution-support functions listed in Appendix D.4, retaining basic execution functions and trajectories. Both conditions use the same development tasks, development model, and resource limits, with three independent development runs per condition. Each resulting frozen harness is evaluated once on the test tasks in Table 1. De-

tailed settings are in Appendix D.4.

Meta-Harness improves performance from both starting harnesses, but the initialized start does not yield a higher final mean (Figure 4). In this comparison, providing these support functions alone is therefore insufficient to improve development outcomes.

T2A’s revision instructions improve results under the same conditions. Holding the initialized start, accessible records, development model, resource limits, and Meta-Harness’s outer evaluation and selection procedure fixed, we replace only the original revision instructions with T2A instructions. These instructions require the development agent to group trajectories by observable features, diagnose problems in light of the distribution of capability bottlenecks, and identify the components or invocation conditions to revise. Before testing, the development agent must specify the expected improvements on target tasks, the performance to preserve on protected tasks, and the evidence needed to establish that the edit was actually triggered. The resulting harnesses’ mean score increases by 21.6 points, and all three development runs score above all three runs with the original instructions. This supports explicitly organizing trajectory diagnosis and revision validation.

Full T2A uses diagnosis to guide subsequent revisions. Meta-Harness with T2A instructions opens a fresh session each round, with the outer program evaluating and selecting candidates. Full T2A preserves continuous context and lets the development agent select validation tasks based on diagnosis and continue revising, increasing the mean score from 52.6 ± 1.9 to 84.6 ± 0.5 . This comparison evaluates the overall development setup, including differences in Claude Code version, reasoning effort, and tool permissions; the gain cannot be attributed solely to context continuity or validation-task selection (Appendix D.4).

5 Conclusion

We presented T2A, which uses execution evidence to diagnose failures, select harness revisions, and validate their effects without updating model weights. Across three environments and three task agents, T2A achieves the highest aggregate score among the evaluated methods. With the starting harness and Meta-Harness outer loop fixed, three independent development runs

per condition show that T2A revision instructions improve the mean ScienceWorld score by 21.6 points. Component ablations link performance gains to reductions in target errors, while development records show how remaining failures guide subsequent revisions. These findings support the view that different task agents and harnesses face different capability bottlenecks, and that the effects of the same intervention vary accordingly. Revision targets should therefore be chosen based on the distribution of deficiencies in the current harness, and checking their effects on capability bottlenecks should guide further revisions.

Limitations

T2A depends on the development agent’s trajectory analysis, interpretations, and implementation choices. Results may vary with its underlying model, retained context, and tools. Updates to Claude Code may change context management, tool use, and execution behavior, so reproducing the harness revisions requires checking its version and configuration.

We ablate only components that can be independently disabled. Other revisions are coupled with the basic execution flow: directly removing them would disrupt normal operation, while replacing them would require corresponding changes to the execution logic. They are therefore excluded from the component ablations in this study.

References

- Lakshya A Agrawal, Shangyin Tan, Dilara Soyly, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. [Gepa: Reflective prompt evolution can outperform reinforcement learning](#). *Preprint*, arXiv:2507.19457.
- Anthropic. 2026a. [Claude Code: Overview](#). Official documentation, accessed October 10, 2026.
- Anthropic. 2026b. [Claude Opus 5 system card](#).
- Anthropic. 2026c. [How Claude Code works](#). Official documentation, accessed October 10, 2026.
- Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. 2026. [Agentrx: Diagnosing ai agent failures from execution trajectories](#). *Preprint*, arXiv:2602.02475.
- Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, Haiming Zheng, and Qing Wang. 2026a. [From failed trajectories to reliable llm agents: Diagnosing and repairing harness flaws](#). *Preprint*, arXiv:2606.06324.
- Mengzhuo Chen, Junjie Wang, Fangwen Mu, Yawen Wang, Zhe Liu, Huanxiang Feng, and Qing Wang. 2026b. [Seeing the whole elephant: A benchmark for failure attribution in llm-based multi-agent systems](#). *Preprint*, arXiv:2604.22708.
- Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, Yuan Xie, Heng Qu, Kun Shao, and Jian Luan. 2026c. [Harnessx: A composable, adaptive, and evolvable agent harness foundry](#). *Preprint*, arXiv:2606.14249.
- Yi Duan, Ying Liu, Zirui Tang, Haodong Chen, Jun Zhou, Yumou Liu, Bangrui Xu, Yukai Wu, Sidi Chen, Yuhan Zhou, Haoyu Wang, Xiaoyou Yu, Shaokun Han, Xuzhou Zhu, Le Zhou, Bolin Lu, Wei Zhou, Jiachen Liu, Nuo Zhou Fang, and 16 others. 2026. [The last ai built by humans: Toward genuine recursive self-improvement](#). *Preprint*, arXiv:2609.11873.
- Shengran Hu, Cong Lu, and Jeff Clune. 2025. [Automated design of agentic systems](#). *Preprint*, arXiv:2408.08435.
- Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. 2026. [Meta-harness: End-to-end optimization of model harnesses](#). *Preprint*, arXiv:2603.28052.
- Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Yida Xue, Xin Xu, Tongtong Wu, Kun Wang, Yang Liu, Zhen Bi, Jungang Lou, Yuchen Eleanor Jiang, and 31 others. 2026. [Skillnet: Create, evaluate, and connect ai skills](#). *Preprint*, arXiv:2603.04448.
- Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. 2026a. [Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses](#). *Preprint*, arXiv:2604.25850.
- Minhua Lin, Juncheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bing He, Zewen Liu, Tianxin Wei, Zongyu Wu, Zhiwei Zhang, Dakuo Wang, Xiang Zhang, Benoit Dumoulin, Cihang Xie, Yuyin Zhou, Suhang Wang, and Hanqing Lu. 2026b. [Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving llm agents](#). *Preprint*, arXiv:2605.30621.
- Haozhe Liu, Tian Ye, Sensen Gao, Qihang Cao, Yitong Li, Mingchen Zhuge, Duomin Wang, Ruihua Zhang, Ping Luo, Jiawang Bian, Lei Zhu, Ligeng Zhu, Enze Xie, and Song Han. 2026. [Sol-pi: Recursively scaling auto-research loops for efficient agent harness](#). *Preprint*, arXiv:2609.20519.

- Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. 2023. [Clin: A continually learning language agent for rapid task adaptation and generalization](#). *Preprint*, arXiv:2310.10134.
- Jun Nie, Yonggang Zhang, Qianshu Cai, Yiu ming Cheung, Xinmei Tian, and Bo Han. 2026. [Evolvenet: Collaborative harness evolution for agent self-improvement](#). *Preprint*, arXiv:2608.04968.
- Qwen Team. 2026. [Qwen3.5: Towards native multi-modal agents](#).
- Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. [Reflexion: Language agents with verbal reinforcement learning](#). *Preprint*, arXiv:2303.11366.
- Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. [Alfworld: Aligning text and embodied environments for interactive learning](#). *Preprint*, arXiv:2010.03768.
- Zihan Tan, Leixin Sun, Zitong Shi, Yitao Liu, Jiajun Wu, Nathaniel Brooks, Jiaru Qian, Xiaoran Shang, Suyuan Huang, Yi Ding, Yangxu Liao, Mukai Li, Qiushi Sun, Shudong Liu, Xuankun Rong, Xiaohang Yu, Zhuo Chen, Hejia Geng, Chenxin Li, and 12 others. 2026. [Metarsi / rsi2: A meta-recursive self-improving system for recursive self-improving systems themselves](#). *Preprint*, arXiv:2609.06396.
- Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjana Balasubramanian. 2024. [Appworld: A controllable world of apps and people for benchmarking interactive coding agents](#). *Preprint*, arXiv:2407.18901.
- Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. [Scienceworld: Is your agent smarter than a 5th grader?](#) *Preprint*, arXiv:2203.07540.
- Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Noah A. Smith, Pradeep Dasigi, and Teng Xiao. 2026. [Rethinking the evaluation of harness evolution for agents](#). *Preprint*, arXiv:2607.12227.
- Peng Xia, Rujun Han, Zifeng Wang, Yanfei Chen, Yufan Zhuang, Yoonho Lee, Chengsong Huang, Han Yu, Zhongying CuiZhu, Yifei Ming, Huaxiu Yao, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. 2026. [Rrsi: Regularized recursive self-improvement of agent harnesses](#). *Preprint*, arXiv:2609.24972.
- Anyi Xu, B Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, CC Yu, and 1 others. 2026. [Deepseek-v4. 1-flash: Pushing the limits of kv cache compression](#). *arXiv preprint arXiv:2609.19969*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. [React: Synergizing reasoning and acting in language models](#). *Preprint*, arXiv:2210.03629.
- Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. 2024. [Self-taught optimizer \(stop\): Recursively self-improving code generation](#). *Preprint*, arXiv:2310.02304.
- Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. 2026a. [Self-harness: Harnesses that improve themselves](#). *Preprint*, arXiv:2606.09498.
- Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. 2026b. [Darwin godel machine: Open-ended evolution of self-improving agents](#). *Preprint*, arXiv:2505.22954.
- Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025. [Aflow: Automating agentic workflow generation](#). *Preprint*, arXiv:2410.10762.
- Xuan Zhang, Wenxuan Zhang, See-Kiong Ng, and Yang Deng. 2026c. [Self-evolving world models for llm agent planning](#). *Preprint*, arXiv:2606.30639.
- Tong Zheng, Xidong Wu, Zheng Zhang, Zhankui He, Chaoyi Zhang, Benjamin Coleman, Ruoqiao Wei, Di Bai, Haolin Liu, Rui Liu, Xue Wang, Yue Zhuan, Wang-Cheng Kang, Renkai Xiang, Heng Huang, Xinwu Cheng, and Yunsong Guo. 2026. [Dreamrsi: Recursive self-improvement through evolving worlds](#). *Preprint*, arXiv:2609.14858.
- Kunlun Zhu, Xuyan Ye, Zhiguang Han, Yuchen Zhao, Bingxuan Li, Weijia Zhang, Muxin Tian, Xiangru Tang, Pan Lu, James Zou, Jiaxuan You, and Heng Ji. 2026. [Agentdebugx: An open-source toolkit for failure observability, attribution, and recovery in llm agents](#). *Preprint*, arXiv:2607.18754.

A Comparison of Revision Procedures

Table 3 compares how related methods turn execution feedback into harness edits and decide which edits to retain. Several methods already use failure diagnosis, scoped edits, and regression checks. The relevant distinction is therefore how evidence is organized, how it determines the scope of a change, and how acceptance is decided.

Relation to the other procedures. The comparison locates T2A’s contribution in organizing diagnosis and validation. It makes the scope of a failure hypothesis explicit before selecting an edit, and keeps unsupported cases unresolved. This

Method	Diagnostic evidence	How edits are organized	Validation and selection
Self-Harness (Zhang et al., 2026a)	Failed trajectories clustered by verifier-grounded failure mechanisms.	Bounded component edits with audit records and passing behaviors to preserve.	Held-in and held-out pass counts must not decrease; at least one must improve.
AHE (Lin et al., 2026a)	Layered trajectory reports and manifests linking evidence, causes, and predicted effects.	Component edits in a decoupled harness workspace.	Next-round attribution checks scores for expected fixes and regressions and supports rollback.
HarnessX (Chen et al., 2026c)	Task briefs and a cross-task outcome overview with edit history.	Edits to typed processors, recorded in change manifests.	A predefined procedure for review, checks, and regression gates.
HarnessFix (Chen et al., 2026a)	Trace intermediate representation links responsible steps to harness-layer flaw records.	Repair operators instantiated with file and behavior constraints.	Repair-scope checks, target-flaw reduction, and regression validation.
EvolveNet (Nie et al., 2026)	Client-specific experience and evidence of gains and regressions.	Aggregate local program adaptations into shared or domain-specific changes.	Commit the integrated harness only if validation gains at least match regressions.
Meta-Harness (Lee et al., 2026)	The proposing agent reads historical code, scores, and execution trajectories.	Free-form code changes chosen by the proposer.	Search-set evaluation and a performance-cost Pareto frontier.
T2A (ours)	Structured execution records provide failure features and error clusters. Hypotheses from representative trajectories are checked against related trajectories to establish concrete execution evidence for locating capability bottlenecks.	Bottleneck diagnosis and test feedback determine changes to content, invocation, or execution, and the scope of those changes.	Guided by bottleneck diagnosis, the development agent judges whether preregistered target-task, protected-task, and edit-triggering conditions are met.

Table 3: How related methods diagnose failures, organize edits, and validate candidates.

complements free-form proposer search, failure-mechanism mining, edit-effect attribution, layer-specific repair, and collaborative program aggregation. Considering the applicability of these procedures to the selected ScienceWorld task set, Section 4.3.3 evaluates T2A development instructions within a fixed Meta-Harness outer loop.

B T2A Workflow

Algorithm 1 summarizes Section 3. Each round groups trajectories below full score, $\mathcal{F}_r$, by observable features into $\mathcal{G}_r$. Candidate diagnoses $\mathcal{Z}_r^{\text{cand}}$ are checked against related successful and failed trajectories to form $\mathcal{Z}_r$ (Equation 3). Groups organize investigation; diagnoses and their applicability conditions guide the edit δ . Every selected task is diagnosed or marked unresolved.

The plan Π_r records target outcomes, protected tasks, edit-triggering checks, and conditions $\mathcal{V}_r$. From current and candidate trajectories, $\mathcal{T}_r^{\text{val}}$ and $\mathcal{T}'_r$, the development agent assigns each condition 1 when met and 0 otherwise; insufficient records count as unmet. Acceptance ($a_r = 1$) requires

Algorithm 1 T2A Workflow

Require: Initial harness H_0 ; development tasks $\mathcal{Q}_r$
Ensure: Accepted harness

- 1: **for** each round r within the experiment limits **do**
- Data collection*
- 2: $\mathcal{T}_r \leftarrow \text{COLLECT}(H_r, \mathcal{Q}_r)$ ▷ Eq. 2
- Harness revision*
- 3: $\mathcal{F}_r \leftarrow \{\tau_d \in \mathcal{T}_r : s_d < s_d^{\text{max}}\}$
- 4: $\mathcal{G}_r \leftarrow \text{GROUP}(\mathcal{F}_r)$
- 5: $\mathcal{Z}_r^{\text{cand}} \leftarrow \text{EXPLAIN}(\mathcal{G}_r, H_r)$
- 6: $\mathcal{Z}_r \leftarrow \text{CHECK}(\mathcal{Z}_r^{\text{cand}}, \mathcal{T}_r, H_r)$
- 7: **repeat**
- 8: $\delta \leftarrow \text{REVISE}(H_r, \mathcal{Z}_r)$ or retain δ
- 9: $H' \leftarrow \delta(H_r)$
- 10: $\Pi_r \leftarrow \text{REGISTER}(H', H_r)$
- 11: $(\mathcal{T}_r^{\text{val}}, \mathcal{T}'_r) \leftarrow \text{VALIDATE}(H', H_r, \Pi_r)$
- 12: $a_r \leftarrow \bigwedge_{v \in \mathcal{V}_r} v(\mathcal{T}'_r, \mathcal{T}_r^{\text{val}})$
- 13: Retain supported findings for further revision
- 14: If $a_r = 0$, revise, revalidate, or withdraw H'
- 15: **until** $a_r = 1$ or H' is withdrawn
- 16: Set $H_{r+1} \leftarrow H'$ if $a_r = 1$; otherwise retain H_r
- 17: **end for**
- 18: **return** Latest accepted harness

all conditions to hold. Otherwise, H_r remains accepted while the candidate is revised, revalidated, or withdrawn.

C Conditional-Score Decomposition

Let B denote the target behavior, $\bar{B}$ its absence, and $J(H) = E_H[s]$ the expected terminal score over tasks and execution randomness under harness H . Let $p_r = P_r(B)$ and $p' = P'(B)$ denote the target behavior’s probability under the current and candidate harnesses. Write $\mu_r^+ = E_r[s \mid B]$ and $\mu_r^- = E_r[s \mid \bar{B}]$, with analogous definitions under the candidate. These conditional expectations require positive probability for the conditioning events. With $\Delta p = p' - p_r$ and $\Delta\mu^\pm = \mu'^\pm - \mu_r^\pm$, the law of total expectation gives

$$J(H_r) = p_r\mu_r^+ + (1 - p_r)\mu_r^-. \quad (C1)$$

Subtracting the current expectation from the candidate expectation gives $\Delta J = J(H') - J(H_r)$:

$$\Delta J = \Delta p(E_r[s \mid B] - E_r[s \mid \bar{B}]) + \Delta_{\text{other}}, \quad (C2)$$

where

$$\Delta_{\text{other}} = p'\Delta\mu^+ + (1 - p')\Delta\mu^-. \quad (C3)$$

The remainder includes changes in subsequent execution and in the composition of the samples conditioned on the target behavior.

D Experimental Settings, Additional Results, and Trajectory Analysis

D.1 Harness Revision Requirements and Baseline Adaptations

Licensing. Our original code is released under the MIT License; third-party materials retain their respective licenses.

T2A initialization and environment adaptation.

T2A initialization augments an existing harness with structured execution records needed for diagnosis and gives the development agent access to modify relevant components and their invocation points. Records should be linked by task, execution step, and model-call identifiers so that the developer can trace the input actually received by the model and its sources, the model’s reply, parsing and action processing, and the action ultimately submitted and its environment feedback. For existing retrieval, memory, or action-checking components, records should also capture invocation outcomes and whether relevant content entered the model input, distinguishing missing content, absent invocation, failed delivery, and noncompliance after delivery.

Appendix D.4 details the ScienceWorld starting harness used in the controlled comparison.

Initial support functions. Table 4 summarizes the recording and execution functions present in each experimental starting harness.

Structured records are per-step records linked by task, step, and model-call identifiers, covering model input and its sources, the reply, parsing and action processing, the submitted action, and environment feedback. *Within-episode memory* is information carried across steps and included in model input. In ScienceWorld, it includes interaction history and object locations confirmed by environment receipts; AppWorld uses interaction history, and ALFWorld uses recent history. *Experience retrieval* places stored experience cards in model input; an empty library retains the mechanism but contains no cards. *Action processing* rewrites or completes a proposed action before execution. In the ScienceWorld start, this includes matching container names to objects in the latest observation and completing mixing steps when the recipe is fully public. *Action checks / execution repair* reject proposed actions with feedback before execution or repair failed execution. These functions are distinct from action processing.

To provide intervention points for guidance and structured action processing, initialization adjusts the flow from model output to environment execution. At the starting stage, some interfaces are not yet equipped with the guidance and checking functions to be developed later, so these flow changes may reduce initial task performance. To maintain comparability of the development interfaces and editable scope, the ablated start retains these flow changes while removing only the additional structured records, object-location memory, and action-processing functions, preserving basic execution functions and action–observation–score trajectories.

WorldEvolver. We adapt WorldEvolver (Zhang et al., 2026c) to build memory on development tasks and freeze it for testing. For comparability with the other methods in Table 1, repeated attempts and memory updates are restricted to training tasks; each test task is attempted once per evaluation. The ALFWorld adaptation uses current observations and does not allow queries for admissible-action lists that are not directly visible to the player.

Function	ScienceWorld	AppWorld	ALFWorld
Structured records	Present	Present	Present
Within-episode memory	Interaction history; confirmed object locations	Interaction history	Recent history
Experience retrieval	Empty card library	None	Empty card library
Action processing	On	None	None
Action checks / execution repair	Off	None	None

Table 4: Support functions in the initial harnesses. Off denotes code that is present but disabled, and none denotes an absent component.

D.2 Experimental Setup and Task Splits

Model execution and scoring. Qwen models (Qwen Team, 2026) use local inference services in all three environments; DeepSeek V4.1 Flash (Xu et al., 2026) is accessed through the official API. The development and reflection model is Claude Opus 5 (Anthropic, 2026b). ScienceWorld retains the last environment score when an output limit or a terminal condition is reached, and reports mean terminal score and fatal-failure rate. ALFWorld reports task completion percentages on `valid_seen` and `valid_unseen`; AppWorld reports task completion percentages on `test_normal` and `test_challenge`.

Software and developer settings. ALFWorld uses version 0.4.2 and AppWorld uses 0.1.3.post1. ScienceWorld uses 1.2.2 for T2A and GEPA and 1.2.3 for minimal prompting, ReAct, and SkillNet. The Meta-Harness baseline and all Meta-Harness arms in Section 4.3.3 use Claude Code 2.1.285 with model identifier `claude-opus-5` and `-effort max`. The ScienceWorld Full T2A launcher uses Claude Code 2.1.284 with the same model identifier and default effort. Its session runs in noninteractive mode, with web access, MCP, and subagents disabled; it uses `-max-turns 1000` and resumes interrupted work in the same session. These launch commands do not explicitly override the developer model’s temperature or `top-p`. The released ALFWorld and AppWorld T2A files do not pin a developer CLI version or effort setting.

T2A task-model decoding and limits. Thinking is disabled for the task models. In ALFWorld, all task models have a 100-interaction limit. The Qwen models use temperature 0.7, `top-p` 0.8, `top-k` 20, and presence penalty 1.5; the per-call output limits are 1,024 tokens for Qwen3.5-27B and 128 for Qwen3.5-9B. DeepSeek V4.1 Flash uses temperature 0.5 and a 512-token output limit. Science-

World uses temperature 0.5, at most 1,200 output tokens per call, 100 environment interactions, and 180 model calls per episode. In AppWorld, the Qwen models use temperature 0, `top-p` 0.8, `top-k` 20, presence penalty 1.5, and a 2,048-token output limit; DeepSeek V4.1 Flash uses temperature 0.5 and a 4,096-token output limit. T2A allows 40 turns per AppWorld task; the ReAct baseline retains its upstream 50-turn limit.

Task splits. For the experiments reported in Table 1, ScienceWorld uses 178 tasks from train for harness revision, selecting seven tasks from each of the 26 task types, or all available tasks when a type has fewer than seven. Its 241-task evaluation subset is selected from test across 26 task types: within each type, we sort by variation ID and take the first 10 variants, or all variants when fewer than 10 are available (Wang et al., 2022).

ALFWorld uses 138 tasks from train for harness revision, with 23 tasks from each of its six task types. Evaluation covers 140 tasks in `valid_seen`, set in rooms seen during training, and 134 tasks in `valid_unseen`, set in unseen rooms (Shridhar et al., 2021).

AppWorld uses train and dev for harness revision. The 168 tasks in `test_normal` use the same application pool as the development splits. The 417 tasks in `test_challenge` each require at least one application held out from those splits, Amazon or Gmail (Trivedi et al., 2024).

For both the experiments reported in Table 1 and the comparison of harness revision instructions, test tasks and their evaluation feedback are not used for harness revision or selection. Harnesses are revised separately for each task agent and frozen before testing.

D.3 Component Ablation Settings

The ablations use the DeepSeek V4.1 Flash ScienceWorld system reported in Table 1 on the same

Component	Full	Removed
Cards	4	28
Check	0	30
Memory	14	48

Table 5: Tasks containing the target error in the full system and after individual component removal. Each row uses its component-specific error definition described in this section.

241 test tasks. Each arm removes one or more component groups while keeping the others unchanged. Combined ablations are reimplemented using the same component definitions. Each ablated harness is evaluated once and paired by task with the first evaluation of the full system among those summarized in Table 1 (mean 85.21), collected in a separate batch. Table 2 reports the means and paired score differences. Target-error counts and trajectory checks below concern the individual-component ablations.

Removing the skill-card library disables card retrieval. Removing action checks bypasses the pre-execution checks for invalid or repeated ineffective actions and task-specific focus prerequisites, together with their blocking feedback. Basic action parsing and environment execution remain unchanged. Removing within-episode memory hides the stored observations and progress from model input; action checks still use it internally. Operating rules are not ablated because removing them would disrupt normal execution and require a manually chosen boundary.

Target-error criteria were defined from records collected during harness revision before this ablation evaluation. For cards, whose design targets procedural and factual knowledge, the reported target error is a fact/answer error identified by blind trajectory reading. For the check component, designed to address erroneous commitments and ineffective repetition, the reported criterion is execution of a focus action that the frozen check rules would block. For memory, the criterion is at least three repeated observations whose results are already known. Counts denote episodes containing the specified error. Newly fatal tasks have a nonfatal full-system outcome and a score of -100 after component removal. Of the 16 newly fatal tasks after removing the check component, 12 reach -100 at the flagged action itself.

D.4 Starting Harnesses, Instructions, and Workflow

Tasks and starting harnesses. The task agent is DeepSeek V4.1 Flash. For comparability, all methods in this comparison receive the same pool of 178 official training tasks spanning 26 types and a budget of 356 development-evaluation episodes after collecting the starting trajectories. This budget excludes the 178 episodes used to collect those trajectories. By comparison, Meta-Harness in Table 1 executes 1,378 development episodes from the minimal-prompt start and reaches a mean score of 66.6. Frozen harnesses are evaluated on the same 241 ScienceWorld test tasks as in Table 1. Training and test tasks are disjoint.

We construct a support-function ablation from the T2A-initialized harness. The initialized start includes the diagnostic records and editable components described in Appendix D.1, together with the object-location memory and action-processing functions of this ScienceWorld implementation. The ablated start removes the additional structured records, object-location memory, and action processing while retaining model calls, action parsing, environment interaction, basic within-episode state, and action–observation–score trajectories. Meta-Harness revises both starting harnesses with the same development tasks, development model, and resource limits. This comparison evaluates the joint contribution of these support functions to the performance of the resulting harnesses.

Functions in the initialized start. Both starts log episode boundaries and each step’s action, observation, and score. The initialized start additionally records model-input components, retrieval candidates, and prompt-token counts per call, together with rejected replies and corrective feedback. Environment-step records include object locations, public facts, the action-processing route, and the action actually submitted. A separate call ledger retains raw model replies; episode summaries record costs and component activity.

Object-location memory stores moves confirmed by environment receipts, including the destination and supporting step. A later action that may move the object without a confirming receipt invalidates the entry. The location table and public facts enter each model prompt.

In chemical and state-change tasks, action processing matches container names to the objects in the latest observation and rewrites the action ac-

cordingly. It can also complete mixing steps when the recipe is fully public. Other action-check and execution-repair components are disabled in this start. The skill-card retrieval and delivery mechanism is present with an empty card library. Object-location memory and action processing are features of this experimental starting harness, not requirements for initializing T2A in every environment.

Replacing the harness revision instructions.

From the same T2A-initialized start, we replace the original Meta-Harness harness revision instructions (Lee et al., 2026) with T2A instructions while retaining its outer evaluation and selection procedure and run settings. T2A instructions require the developer to group trajectories, propose and check explanations in context, and validate revisions against predefined conditions. Each condition has three independent development runs.

Full T2A harness revision. Full T2A uses the T2A-initialized start in a continuous session with the development agent. It receives starting trajectories for 178 tasks, follows the T2A workflow to select validation tasks and repeat counts, and continues revising the harness based on diagnosis and validation results. Full T2A also has three independent development runs. Unlike the instruction-replacement comparison, which holds the developer settings fixed, the Full T2A comparison also differs in Claude Code version, effort, and tool permissions as described in Appendix D.2. It therefore evaluates the combined development setup.

Selection, stopping, and context. In Full T2A, the development agent writes and judges its validation conditions from execution records without researcher review. The program enforces the budget, records workspace fingerprints, and freezes submissions; it does not independently judge whether the semantic acceptance conditions hold. The last explicitly submitted harness is evaluated; if there is no submission, the starting harness is used. Full T2A uses one continuous development session and may finish at the agent’s discretion. Exhausting the episode budget prevents further environment runs but still permits review of existing evidence and submission. Context is managed by Claude Code’s built-in compaction (Anthropic, 2026c), without a separate context-recovery mechanism. No separate limit is imposed on inner revision attempts; environment runs remain subject to

the total episode budget. No per-round candidate-proposal time limit is imposed. By contrast, all Meta-Harness arms use up to five rounds, each with a fresh session and a 2,400-second proposal limit. A round that produces no candidate is not rerun. The outer program evaluates submitted candidates and selects the highest search-set mean, including the starting harness; ties favor the earlier candidate. This selection rule also applies to Meta-Harness with T2A instructions.

Evaluation settings. All arms use DeepSeek V4.1 Flash at temperature 0.5 with thinking disabled, at most 1,200 output tokens per call, 100 environment interactions, and 180 model calls per episode. These task-execution limits are enforced outside the editable harness. Scores are recorded for all 241 tasks. Each of the three harnesses from independent development runs is evaluated once. When the execution budget is exhausted or a terminal condition is reached, the last available environment score is retained.

D.5 Failure Classification and Trajectory Analysis

Category definitions and denominators. Figure 3 uses one evaluation per harness. Each cell is a count divided by the size of the environment’s whole test set; all three blocks share the same denominator. The intermediate version is the accepted harness snapshot nearest the middle of each harness revision sequence. For ScienceWorld–DeepSeek V4.1 Flash, this is the snapshot at development episode 190.

ScienceWorld ($n = 241$ episodes). Each cell counts point-losing episodes ($s < 100$) exhibiting the phenomenon; an episode may receive several labels.

- *Format*: at least 10% of model replies contain no executable action.
- *Action syntax*: at least 10% of submitted actions receive “No known action matches that input.”
- *Loop / memory*: ten consecutive steps without a score change use at most three distinct actions, none of them wait.
- *Procedure*: points are lost because required steps are not carried out in a workable order.
- *Fact / answer*: a committed answer is wrong because of an incorrect fact or identification.

- *Fatal focus*: final score is -100 .
- *Missed focus*: no focus action is accepted during the episode.

Procedure and Fact / answer are labelled by Claude Opus 5 (Anthropic, 2026b) using anonymized, shuffled trajectories without system identities; the other five categories are computed mechanically.

AppWorld ($n = 585$ tasks). Each failed task receives one primary cause. Rules are checked in a fixed order and the first match wins, so an earlier cause can mask a later one.

- *Output protocol*: parser-rejected replies, truncated output, misuse of the answer slot, or results never printed.
- *Auth*: requests rejected as unauthorized, with no successful app call.
- *API usage*: turns spent on unknown APIs or invalid arguments.
- *Execution*: turn limit, identical code resubmitted, context budget, or infrastructure error.
- *No action*: the required change is never made.
- *Missing items, Extra items, Side effects, and Wrong edit*: a change is made, but items are omitted, extra items included, unrelated records changed, or a field value or selected object is wrong, respectively.
- *Wrong answer*: only the returned answer value is wrong.

ALFWorld ($n = 274$ tasks). For each harness, its trajectories are classified independently: each failed task is assigned to the first task milestone it never reaches.

- *No interaction*: fewer than three steps, or most replies rejected.
- *Not found*: the target object never appears.
- *Seen, not taken*: the target appears but is never picked up.
- *Not processed*: the required clean, heat, cool, or light step is not completed.
- *Not placed*: the object never reaches the target receptacle.
- *One of two*: only one object is placed in a two-object task.

ReAct on ScienceWorld with Qwen3.5-9B.

We read paired trajectories from one minimal-prompt evaluation and one ReAct evaluation on the same 241 tasks, using the first evaluation of each harness among those summarized in Table 1. Under the definitions above, the percentage of episodes with format errors falls from 13% to 3%, while fatal failures rise from 29% to 62%. Among the additional failures under ReAct, 34.1% contain behavior that contradicts descriptions already present in the model’s prompt. In 61.9%, ReAct makes task progress earlier than minimal prompting but later executes a fatal action despite having more steps remaining. These trajectories suggest that improving one kind of execution problem can shift the capability bottleneck: the model makes progress earlier, yet still fails because of a later incorrect decision.

E Historical Diagnosis and Revision Cases

E.1 Successive Bottlenecks in ALFWorld with DeepSeek V4.1 Flash

This case follows diagnosis, revision, and validation on ALFWorld training tasks with DeepSeek V4.1 Flash.

From delivered guidance to action checks. In one failed task, the goal required a mug, but the task agent took and cooled a cup and failed after 100 steps. The exact-type skill card had been retrieved and included in the prompt. Inspection of 13 failed episodes identified nine in which delivered guidance was not followed. The developer therefore converted the type rule into a pre-execution action check: actions selecting the wrong object type were rejected, and the rejection reason entered the next model call.

A counterexample exposed an error in this check. In a look-at task, a lamp serves as a tool; rejecting its pickup because it does not match the target object’s type blocks a valid action. The revised check distinguished task objects, tools, and containers. On the same 120 training tasks, completion increased from 85 under minimal prompting to 98 under the candidate. However, successful two-object tasks fell from 17 to seven, and 17 tasks that minimal prompting completed now failed.

Diagnosing the remaining failures. The developer read the 17 regressions and all 22 candidate failures. In 16 regressions, the first error in-

Task situation	Old trajectory	Revised trajectory	What the trajectories establish
Prepare an intermediate color for tertiary mixing	Repeated kitchen search without finding materials; final score 0	Enter the art studio, mix blue/yellow paint; environment confirms green at 23 points; then select the wooden cup as the target and end at -100	Material location and preparation improve, but target reference remains faulty; lower final score obscures earlier progress
Compare inclined-plane angles	Search/navigation without a descent experiment; final score 10	Complete the first descent and read time; while preparing the second, use focus on the stopwatch as a reset and end at -100	First experiment progresses, but using focus to reset the stopwatch ends the task
Infer dominant/recessive plant traits	Sow seeds but fail to obtain water effectively; plants do not grow; final score 16	Grow two parent plants to flowering, but answer from parental leaf shape instead of offspring evidence; end at -100	Care operations progress, but genetic evidence remains insufficient

Table 6: Historical same-task cases with local progress and subsequent fatal failure, evaluated under the execution limits in effect when each harness version was evaluated.

volved an action being allowed despite an unmet observable prerequisite: moving toward a target that was not visible, taking a second object before putting down the first, or retrieving an object already placed correctly. The trajectories did not support the earlier explanation that search feedback caused the regressions, so that explanation was withdrawn.

The next revision added checks for visibility, whether the agent was already holding an object, and whether an object had already been placed correctly. Implementation review also found that opening a container was incorrectly treated as a room change; this state-tracking error was corrected before validation.

Validating behavior and selecting revisions. A 2×2 comparison independently enabled or disabled the revised action checks and the new card library on 32 training tasks, with two evaluations per condition. Enabling the checks without the new cards increased completion from 21 to 31 in both evaluations. Adding the new cards on top of the checks changed completion from 31 to 31 and from 31 to 30. The checks were accepted, and the new card library was withdrawn.

The trajectories also showed changes in the targeted behavior. Visibility checks rejected 186 actions; in 168 cases, the next action switched to navigation or search. Checks against taking another object while already holding one rejected 59 actions; 35 were followed by putting down the held object. The check against retrieving an already placed object had only one opportunity to intervene; it rejected the action, and that episode suc-

ceeded. In a subsequent evaluation on all 120 training tasks, the frozen harness completed 118. This sequence shows how validation exposed a new bottleneck after the first revision and guided the next revision and acceptance decision.

E.2 Reward Structure Can Obscure Local Progress

We revisit historical paired outcomes on the same 184 tasks, including 31 transitions from nonfatal endings to fatal failures. After excluding nine tasks that already had full scores under the old harness, we read complete old/new trajectories for the remaining 22, distinguishing operational progress, experimental evidence, and fatal submissions. Table 6 gives three representative situations.

E.3 A Diagnosis-to-Revision Example: Life-Span Ranking

This example summarizes a Full T2A development record in ScienceWorld with DeepSeek V4.1 Flash, showing a shift from skill-card content to control logic.

Triage and explanation checks. A script selected 118 point-losing trajectories from the 178 starting trajectories. Of the 27 trajectories ending at -100, 24 ended with a focus on action. This pattern motivated the explanation that focus submits an answer and that selecting a present but incorrect object can cause fatal failure. Counterexamples narrowed the explanation: in measure-melting-point-known-substance-v0157, two focuses on the same object at steps 12 and 14 were nonfatal; in

use-thermometer-v0125, a focus on an object absent from the room at step 36 was rejected by the parser without fatal failure. Thus neither repeated focus nor parser rejection alone implied a fatal outcome.

Card revision and regression. The first revision added life-span cards with typical species life spans and stated that an egg counts as an animal of its species. Two target tasks improved from -100 to 100 , but the protected task `lifespan-shortest-lived-v0056` fell from 100 to -100 : the model chose a frog egg (six years) over a baby mouse (four years). The developer neutralized the wording and added a worked example. In a later round, the same task again ended at -100 despite that exact example being in the card. The developer concluded that prose did not reliably support the comparison and shifted the revision from experience content to control logic.

Explicit ranking and registered checks. The revised harness computes a ranking of animals observed in the episode and inserts it into the model request as `animal_life_spans`. A separate check blocks focus on a non-extreme animal when a strictly better candidate is known; it does not intervene for species without a recorded life span. This candidate, denoted R5 in the development record, also included a plant-stage check, stale-record cleanup, and a rule that stops yielding after three repetitions of the same state. Before validation, the developer registered the following life-span checks: the target task `v0056` should score 100 ; each of five other, protected life-span tasks should remain at 100 ; and the ranking table should be present in the requests for all six episodes, agree with an offline recomputation from raw observations, and place every accepted focus at the extreme requested by the task.

Outcomes and candidate disposition. The target and all five protected life-span tasks scored 100 . The ranking table was present in all six episodes and matched the offline recomputation. The life-span blocking rule never fired because the model selected the correct answers after receiving the table; the record marked it as not triggered rather than failed. The registered plan also required at least one of the two new blocking rules to fire, with trajectory checks distinguishing non-triggering from failure for the other rule.

The whole R5 configuration was retained as a

candidate but was not accepted: neither new blocking rule fired, and two protected tasks from other families regressed. Under the preregistered fallback, the higher-scoring configuration remained a candidate for further validation. R6 evaluated the same configuration without code changes on 52 previously unrun variants under a new preregistered plan and accepted it after all conditions passed. The plant-stage check blocked one incorrect answer. The life-span check remained untriggered because all six life-span tasks were answered correctly; its blocking effect therefore remained unverified. Acceptance applied to the whole configuration, not to independent verification of every rule it contained.

F Use of AI Assistants

We used AI assistants for code implementation and debugging, literature search, analysis support, and manuscript drafting, revision, and translation. The authors are responsible for the final manuscript, code, and reported results.