StarHarness: Evolving Harnesses with Stratified Search for Enterprise Environments

Esakkivel Esakkiraja[1], **Denis Akhiyarov**[1], **Vikas Yadav**[1], **Sai Rajeswar**[1,2,3], **Patrice Bechard**[1], **Sridhar Nemala**[1], **Sagar Davasam**[1]

[1]ServiceNow, [2]Mila, [3]Université de Montréal

We present StarHarness, a framework for evolving environment-specific agent harnesses while keeping model weights fixed. The evolved harness can include prompt and task framing, tool interfaces, skills, MCP-backed providers, subagent structure, and agent-loop configuration. StarHarness constructs a compact evolution pool by stratifying tasks by baseline failure behavior, separates proposer-visible search tasks from proposer-hidden selection tasks, and reserves held-out tasks for evaluating generalization. Across ITBench SRE, EnterpriseOps-Gym ITSM, and AutomationBench Finance, harness evolution improves full-benchmark performance by 20–35 percentage points over the default harness, after 4–12 accepted changes per environment. These gains persist on tasks excluded from evolution and transfer without re-evolution across GPT and Qwen model families. Trace analysis links them to interface repairs, environment conventions, and operational knowledge that compresses search, with fewer false-positive diagnoses and shorter trajectories in several settings. StarHarness therefore offers a practical way to reduce persistent model–environment mismatch in tool-rich enterprise tasks.

Correspondence: esakkivel.esakkiraja@servicenow.com, denis.akhiyarov@servicenow.com

: https://github.com/ServiceNow/StarHarness

servicenow

1 Introduction

Modern LLM agents act through harnesses that define how they use tools and interpret state. Tool-augmented agents depend on the interaction between reasoning, actions, and external state (Yao et al., 2023; Schick et al., 2023). In a tool-rich enterprise task, harness design determines whether the agent can retrieve the right records, perform valid mutations, and satisfy exact final-state checks (Rombaut, 2026; Meng et al., 2026). Interface design can change agent behavior and task success without changing model weights (Yang et al., 2024).

Enterprise service-management and workflow agents must act through stateful backends, large tool surfaces, cross-step dependencies, and domain conventions that tool schemas often omit. Our benchmarks capture different instances of this broader problem: ITBench tests root-cause analysis over operational telemetry (Jha et al., 2025); EnterpriseOps-Gym tests state-changing ITSM workflows against database assertions (Malay et al., 2026); and AutomationBench tests multi-application finance workflows with programmatic state checks (Shepard & Salimans, 2026). Together, they expose the state dependencies, policy constraints, and final-state requirements that make enterprise tool use difficult (Yao et al., 2024; Lu et al., 2024). They raise three questions: Can search evolve a harness from a compact task subset without overfitting to that subset? Do the resulting changes transfer to unseen tasks and other models? Which interaction failures does harness evolution repair?

StarHarness addresses these questions by evolving environment-specific harnesses while keeping model weights fixed. The method stratifies tasks by baseline failure behavior, separates proposer-visible search tasks from proposer-hidden selection tasks, and reserves held-out tasks for evaluation.

Our contributions:

1. **Efficient stratified harness evolution.** We introduce a harness-evolution protocol that searches over compact, failure-mode-stratified task subsets while separating proposer-visible search tasks, proposer-hidden selection tasks, and held-out evaluation. This provides a direct measure of generalization.

2. **Task and model transfer of specialized harnesses.** Across three stateful enterprise benchmarks, harnesses evolved with one model improve tasks excluded from evolution and transfer without re-evolution across GPT and Qwen models.

3. **Analysis of learned specialization.** We identify three recurring forms of environment specialization: interface repair, environment conventions, and operational knowledge that compresses search. We connect them to changes in agent precision, convergence, and efficiency.

2 Related Work

2.1 Prompt and Harness Optimization

Prompt optimization methods search instructions and demonstrations with generated candidates, textual feedback, or evolutionary selection (Opsahl-Ong et al., 2024; Agrawal et al., 2025). Agent-architecture methods extend the search to executable modules and workflow structure (Khattab et al., 2024; Zhang et al., 2025).

Recent harness-level systems edit executable scaffolding or co-evolve harnesses with model policies and weights (Lee et al., 2026; Hebbar et al., 2026).

StarHarness studies a narrower deployment problem: adapting a frozen model's harness to a stateful enterprise environment. Its search space includes prompts, tools, skills, MCP providers, subagents, and execution policy. Evaluation protocols differ across this literature: Meta-Harness uses held-out sets for text classification and mathematical reasoning, but searches and reports final TerminalBench-2 performance on the same 89-task benchmark, which its authors frame as a discovery setting (Lee et al., 2026). We use task-level separation, proposer-hidden selection, and held-out evaluation to distinguish search performance from generalization, following recent calls for stricter harness-evolution evaluation (Wang et al., 2026). The resulting edits remain external to model weights and can be tested and reverted as ordinary code changes.

2.2 Agent Benchmarks and Enterprise Environments

Existing benchmarks cover related agent settings with different interaction interfaces. WorkArena (Drouin et al., 2024) studies browser-based knowledge work on the ServiceNow platform, while Terminal-Bench 2.0 (Merrill et al., 2026) evaluates difficult command-line tasks in isolated environments. We focus on three complementary enterprise settings: ITBench (Jha et al., 2025) contains 40 Kubernetes root-cause analysis scenarios in which the agent inspects alerts, events, traces, metrics, and topology before producing a structured diagnosis; EnterpriseOps-Gym (Malay et al., 2026) contains 103 ITSM workflows spanning incident, problem, change, knowledge, and user-management tasks, with SQL verifiers checking the resulting ServiceNow state; and AutomationBench (Shepard & Salimans, 2026) contains 100 finance workflows across 47 simulated SaaS applications, scored by programmatic assertions over environment state. These benchmarks require agents to operate domain tools, maintain persistent state, and satisfy workflow-specific objectives. We use them to measure task-level generalization and transfer of a frozen specialized harness across models.

3 Method

3.1 Overview

We define **harness evolution** as outer-loop optimization of the executable scaffold surrounding a fixed language model. In our implementation, the **StarHarness optimizer** is a coding harness built on Oh My Pi (contributors, 2026), a variant of the Pi agent harness (Zechner, 2026). It hosts the proposer and executes the edit, validation, and evaluation loop: it exposes the permitted repository and benchmark traces, applies a candidate patch, runs checks, and launches benchmark evaluations. The optimizer modifies the separate **Stirrup harness** (Artificial Analysis, 2026), which serves as the agent harness under evaluation. Its editable surface includes prompt and task framing, tool definitions and schemas, argument preprocessing, skills, MCP providers, subagent structure, context management, verification, and finish logic. The model weights and benchmark remain fixed during evolution.

Let $\mathcal{D}$ denote a benchmark, h a harness, M the agent model, and $J(h;\mathcal{D})$ the cost function, defined here as the mean task score obtained by running M with h on $\mathcal{D}$; higher values are better. The target is

$$h^* = \arg\max_{h \in \mathcal{H}} J(h; \mathcal{D}_{\text{holdout}}), \tag{1}$$

where $\mathcal{H}$ is the permitted harness space. StarHarness cannot access held-out outcomes during search, so it approximates this target with proposer-visible search tasks and proposer-hidden selection tasks where applicable, reserving $\mathcal{D}_{\text{holdout}}$ for final evaluation.

The coding harness runs three stages: proposal, validation, and evaluation. The proposer reads the current harness and search-set traces and returns a candidate patch. The validator checks scope, imports, and a single-task smoke test. The evaluator runs valid candidates on the hidden selection set when one exists; the evolution loop applies the fixed acceptance rule. Following the autoresearch pattern (Karpathy, 2026), the evolution loop measures a baseline, proposes one bounded intervention, evaluates it, retains only an improvement, and repeats from the resulting frontier. A persistent memory ledger links patch, session, and evaluation artifacts and carries frontier scores, per-task outcomes, accepted hypotheses, and discarded attempts into later iterations. For tree search, the ledger is a solution journal containing candidate nodes, parent links, scores, failure states, and promotion decisions. Figure 1 gives the procedure in algorithmic form.

The evolution loop first runs a proposer-selected single-task *test flip*. If the candidate does not flip that task, the loop records a rejection and skips the expensive evaluation. A candidate that passes the gate is evaluated on the selection set; promotion remains deterministic because the candidate must improve the selection mean, with verifier rate used as a tie-breaker only when that additional metric is available. The loop commits accepted candidates as the new frontier and refreshes the search traces. It reverts rejected, invalid, and crashed candidates to the prior frontier.

3.2 Task Partition and Stratified Sampling

We construct the task partition before evolution. From the full benchmark of N tasks, we first retain N' tasks with reproducible evaluation. We then sample an evolution pool of K tasks, typically $K \approx N/2$, using three baseline descriptors:

- **Baseline failure mode** (e.g., wrong_tool, context_loss, missing_evidence, premature_conclusion)
- **Baseline task score**
- **Verifier pass rate**

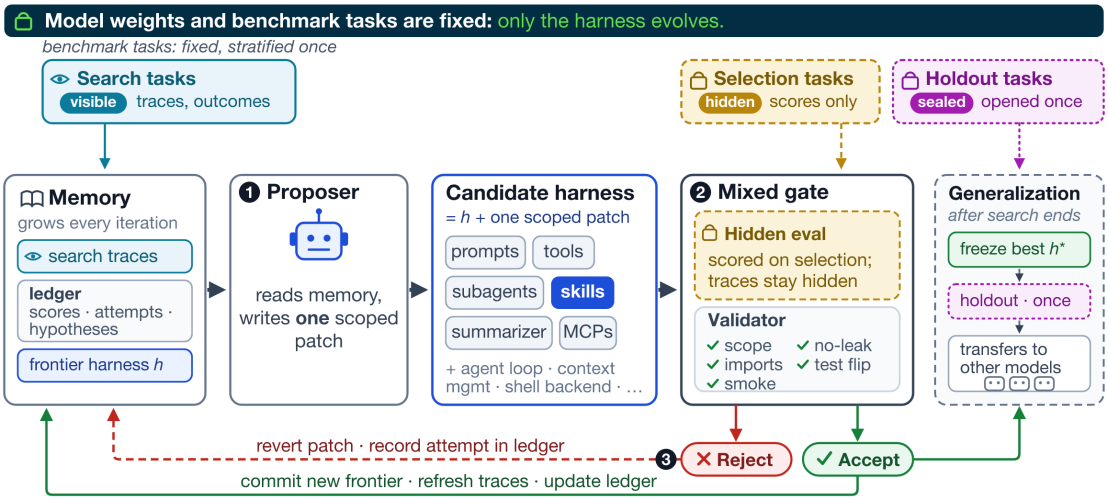
We compute these descriptors from a baseline run on all N' tasks before creating the partition. When the protocol uses a selection set, we split the evolution pool into proposer-visible search tasks and proposer-hidden selection tasks while matching their baseline score, failure-mode, and verifier-pass distributions. The proposer receives search-task traces and outcomes, but no selection-task contents, traces, verifier feedback, or per-task outcomes. The remaining $N' - K$ tasks form the holdout set and never affect proposal or acceptance.

3.3 Search Strategies: Exploration and Exploitation

We use the same proposer, validator, evaluator, and acceptance score in two search procedures. In **hill climbing**, the state is a single frontier harness. At iteration t, the proposer emits one patch Δ_t from traces of the current frontier; the system keeps $h_t \oplus \Delta_t$ if and only if it strictly improves the selection score, or ties it while improving an available verifier metric, and otherwise restores h_t.

In **tree search**, the state is a set of candidate nodes. Each node stores a parent pointer, cumulative patch, search traces, validation status, and selection score. The proposer can *explore* a failure pattern, *draft* a patch, *debug* a failed candidate, *merge* two compatible nodes, or *improve* an existing node. Valid nodes are scored on the same hidden selection set; the best surviving node becomes the frontier for subsequent refinement. This preserves alternative hypotheses instead of committing after the first accepted edit.

We use these modes as an exploration–exploitation control experiment on EnterpriseOps-Gym alone. Tree search explores alternative hypotheses; hill climbing then exploits the best tree frontier through bounded local



Algorithm 1: StarHarness evolution

Input: benchmark $\mathcal{D}$; fixed model M; seed harness h_0; proposer P; budget B
Output: evolved harness h^*

Run h_0 on $\mathcal{D}$; record scores and failure strata
Construct $D_{\text{search}}, D_{\text{select}}, D_{\text{holdout}}$
$h \leftarrow h_0$; $s \leftarrow J(h; D_{\text{select}})$; initialize ledger L
for $t = 1, \ldots, B$ **do**
  $L \leftarrow$ load ledger; $T \leftarrow$ gather search traces
  $\Delta \leftarrow P(h, T, L)$; capture Δ as a scoped patch
  if scope, leakage, import, or smoke check fails: revert and record rejection
  else run proposer-selected test flip
    if test flip fails: revert and record rejection
    else evaluate $h \oplus \Delta$ on hidden D_{select}
      if selection score improves, or ties with an available verifier metric that improves: commit $h \oplus \Delta$; update L; refresh T
      else: revert Δ; update L
return h; evaluate once on D_{holdout} and $\mathcal{D}$

Figure 1 StarHarness evolution. *Top:* the corresponding workflow. *Bottom:* the executable procedure. The proposer reads the current frontier, persistent memory ledger, and search-set traces, but never selection or holdout traces. Candidates pass scoped validation and a proposer-selected test flip before hidden selection evaluation; accepted edits advance the committed frontier and refresh the ledger and search traces. For tree search, the ledger stores candidate nodes and parent links rather than only one greedy frontier.

edits. The stages are sequential, so this design describes how the modes complement each other rather than providing a causal head-to-head comparison.

3.4 Guardrails and Candidate Isolation

StarHarness performs benchmark-assisted environment adaptation: the proposer may inspect search-task trajectories and their evaluation outcomes to diagnose recurring failures, but guardrails prevent it from encoding task-specific solutions.

Each candidate is a git diff against the current frontier, scoped to the benchmark's editable directories and the shared agent framework. Forbidden changes include:

- Branching on task IDs or hard-coded answers
- Verifier or assertion content in agent prompts
- Ground-truth tables or hidden-state access
- Benchmark-specific answer mappings

These constraints target reusable environment behavior rather than individual task solutions. The validator checks scope, imports, and a single-task smoke test before benchmarking. The evolution loop reverts candidates

that fail any check without running the selection evaluation.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate on three enterprise benchmarks:

- **ITBench SRE** (Jha et al., 2025): the latest public release of the ITBench-AA SRE set, consisting of 40 Kubernetes root-cause analysis scenarios from the OpenTelemetry demo application [1]. The agent inspects an offline incident snapshot containing alerts, events, traces, metrics, and topology, then writes a structured diagnosis identifying the contributing entities. Our Stirrup baseline follows the documented setup where available, including the sandboxed code-execution environment, shell-based snapshot inspection, and structured JSON output.
- **EnterpriseOps-Gym ITSM** (Malay et al., 2026): 103 ITSM oracle tasks (incident, problem, change, knowledge, and user management), a subset of the broader 1,150-task benchmark, against a ServiceNow MCP backend. Graded by SQL verifiers checking final database state.
- **AutomationBench Finance** (Shepard & Salimans, 2026): 100 finance workflow tasks (AP/AR, expenses, reporting, bookkeeping) across 47 simulated SaaS applications. Graded by programmatic assertions on environment state. We report the share of the domain's objectives the model achieved; a guardrail violation assigns the task a score of zero. Our Finance-100 subset and Stirrup agent harness differ from the benchmark paper's default setup, so scores are not directly comparable to those reported by Shepard & Salimans (2026).

Across all three benchmarks, we score following the Artificial Analysis evaluation description [2]. For AutomationBench, this score is computed from the share of the domain's objectives the model achieved, with a guardrail violation assigning the task a score of zero.

Models. We evolve harnesses using GPT-5.4 (medium reasoning) as the agent under test and GPT-5.4 as the proposer running inside the Pi-based coding harness (OpenAI, 2026). We then evaluate the same frozen evolved harness, without re-evolution, on additional GPT and Qwen models, including Qwen3.6 (Qwen Team, 2026), for each benchmark.

Baselines. For each benchmark, the baseline is the unmodified Stirrup agent framework with default prompts and tool configurations. We additionally compare against the standalone Pi and Codex harnesses and GEPA prompt optimization applied on top of the Pi harness.

4.2 Main Results

Figure 2 summarizes the full-benchmark harness comparison. Scores use each benchmark's metric; higher is better. StarHarness (Stirrup) is the evolved Stirrup harness, while GEPA (Pi) denotes prompt optimization applied on top of the Pi harness.

StarHarness (Stirrup) is the strongest configuration on all three benchmarks. Relative to GEPA (Pi) (Agrawal et al., 2025), the gains are +13.8, +22.3, and +17.6 percentage points on ITBench, EnterpriseOps-Gym, and AutomationBench, respectively. The comparison is descriptive: the systems differ in prompts, tools, execution policies, and harness architecture, so it does not isolate a single causal component. The result shows that environment-specific harness design can add substantial performance beyond prompt optimization alone.

The performance gains also coincide with lower estimated GPT-5.4 inference cost per task: StarHarness reduces cost by 17% on ITBench, 53% on EnterpriseOps-Gym, and 29% on AutomationBench at published rates (OpenAI, 2026).

[1] https://huggingface.co/datasets/ArtificialAnalysis/ITBench-AA
[2] https://artificialanalysis.ai/evaluations/

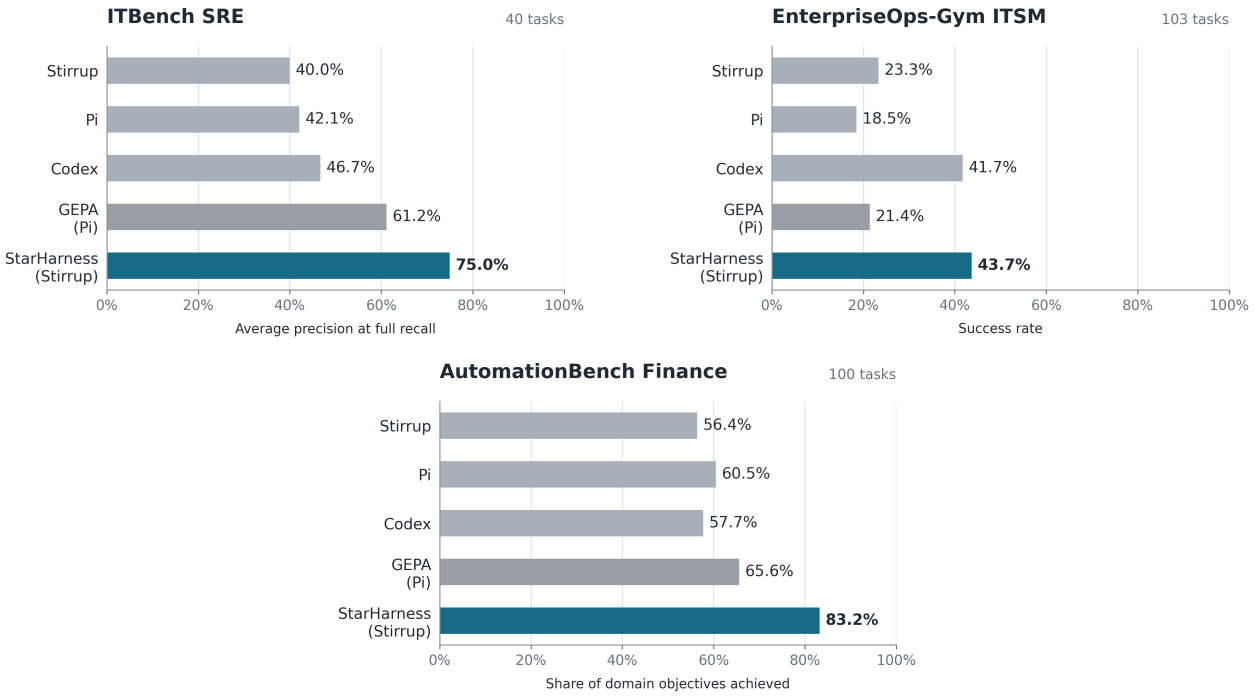


Figure 2 Full-benchmark harness comparisons for ITBench SRE, EnterpriseOps-Gym ITSM, and AutomationBench Finance. GEPA (Pi) denotes prompt optimization applied on top of the Pi harness; StarHarness (Stirrup) denotes the evolved Stirrup harness. The AutomationBench score is the share of the domain's objectives the model achieved, with a guardrail violation assigning the task a score of zero.

4.3 Frozen Cross-Model Transfer

Table 1 consolidates frozen-harness transfer results across all three benchmarks. Each evolved-harness score uses the same StarHarness artifact evolved with GPT-5.4; no benchmark-specific re-evolution was performed for the transferred models. The harness improves every transferred model in the table, including models from both GPT and Qwen families.

Table 1 Frozen StarHarness transfer across models. Values are full-benchmark scores; parentheses give reasoning level when applicable.

Benchmark	Model	Baseline	StarHarness	Δ
ITBench	Qwen3.5-27B	25.6%	70.0%	+44.4 pp
ITBench	GPT-5.4-mini (medium)	33.1%	79.4%	+46.3 pp
ITBench	GPT-5.4 (medium)	40.0%	75.0%	+35.0 pp
ITBench	GPT-5.5 (medium)	50.8%	78.7%	+27.9 pp
EnterpriseOps-Gym	Qwen3.6-27B	18.2%	38.8%	+20.6 pp
EnterpriseOps-Gym	GPT-5.4-mini (medium)	13.6%	31.1%	+17.5 pp
EnterpriseOps-Gym	GPT-5.4 (medium)	23.3%	43.7%	+20.4 pp
EnterpriseOps-Gym	GPT-5.5 (high)	37.8%	48.5%	+10.7 pp
AutomationBench	Qwen3.6-27B	48.2%	75.5%	+27.3 pp
AutomationBench	GPT-5.4-mini (medium)	29.6%	70.0%	+40.4 pp
AutomationBench	GPT-5.4 (medium)	57.1%	83.2%	+26.1 pp
AutomationBench	GPT-5.5 (medium)	59.6%	84.9%	+25.3 pp

For context, the EnterpriseOps-Gym run also reports external Claude reference scores of 48.1% (Fable 5), 35.9% (Sonnet 5), and 35.5% (Opus 4.8 max), shown in Figure 3. These reference runs do not use the controlled baseline–StarHarness comparison.

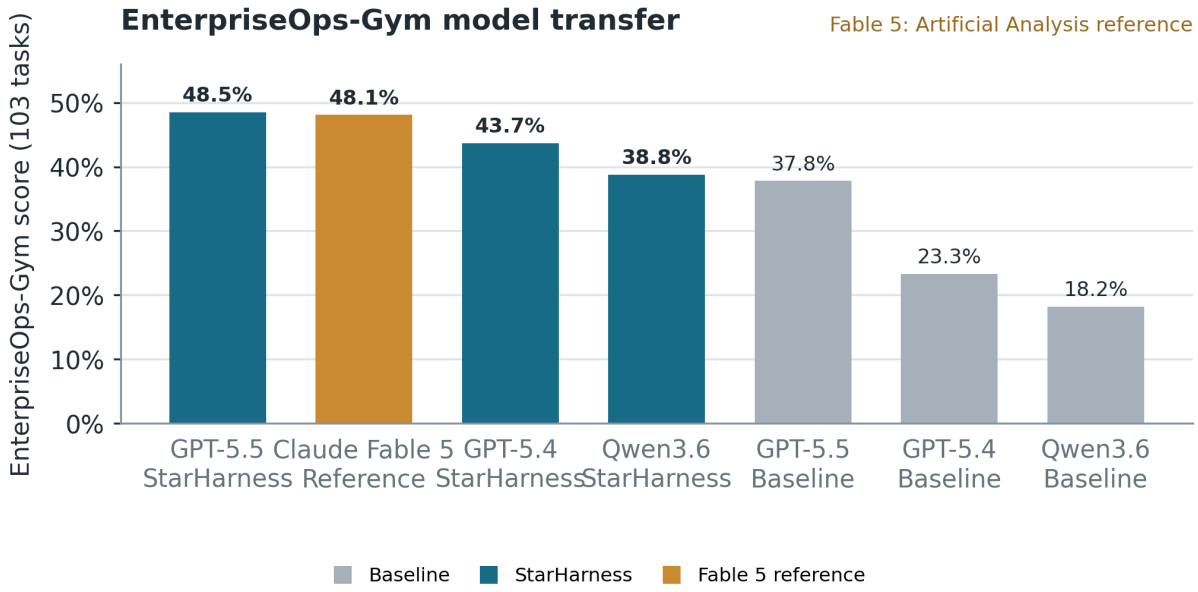


Figure 3 EnterpriseOps-Gym model-transfer leaderboard. The Claude Fable 5 score is an external Artificial Analysis reference and is not part of the controlled baseline–StarHarness comparison.

4.4 Held-Out Generalization

Table 2 summarizes GPT-5.4 generalization across the three benchmarks.

Table 2 GPT-5.4 generalization summary. Values are absolute gains in percentage points.

Benchmark	Model	Evolution set	Held-out
ITBench	GPT-5.4	+45.0	+31.7
EnterpriseOps-Gym	GPT-5.4	+22.0	+15.1
AutomationBench	GPT-5.4	+23.0	+29.3

5 Analysis: What Harness Evolution Learns

Across the three evolution runs, 21 patches were accepted: 4 for ITBench, 12 for EnterpriseOps-Gym, and 5 for AutomationBench. On EnterpriseOps-Gym, 8 accepted patches came from the tree-search exploration stage and 4 from the subsequent hill-climbing stage. The tree stage exposed interacting schema, prompt, and tool failures; hill climbing then added targeted linkage and grounding fixes. Across benchmarks, the edits addressed three kinds of model–environment friction.

Interface repair. EnterpriseOps-Gym evolution repaired MCP argument handling, preserved compound schemas, pruned misleading fields, and added linkage and self-reference cues. These changes made the existing environment interface more usable without changing the task data or verifier. AutomationBench similarly gained structured row operations that replaced fragile raw spreadsheet edits.

Codex provides a close reference on EnterpriseOps-Gym: it scores 41.7%, compared with 43.7% for StarHarness. Its default MCP preprocessing normalizes and compacts schemas, reducing invalid calls to the strict Docker-backed server, which rejects explicit `null`s and empty values when they violate the schema. StarHarness learned complementary repairs: it narrows and enriches schemas, then strips null, empty, and placeholder arguments before calls reach the server.

Environment conventions. The evolved harness made implicit operational rules explicit. Examples include the EnterpriseOps-Gym execution contract, coupled priority and impact/urgency updates, and the requirement

to preserve relationship fields. In AutomationBench, system guidance anchored relative dates to the sandbox clock and added a triage step before mutations. The harness encoded these procedures while the model weights stayed fixed.

Operational knowledge and search compression. Several patches moved repeatable work out of open-ended reasoning. ITBench gained a forensics overview that ranks candidate upstream causes from observed evidence. AutomationBench gained date and finance calculators for deterministic temporal and numerical operations. These changes encode operational knowledge and compress search; they expose derived information from the task environment without accessing labels or verifier state.

5.1 Trajectory Analysis

Each table reports the baseline, evolved harness, and absolute change for the available paired records.

ITBench. Table 3 reports full-benchmark score and execution traces over all 40 tasks. Evolution increased task score from 40.0% to 75.0%, reduced turns and false positives, and increased true positives while using a comparable number of shell calls.

Table 3 ITBench (GPT-5.4) full-benchmark score, execution-trace, and estimated API-cost comparison over all 40 tasks.

Metric	Baseline	Evolved	Δ
Full-benchmark task score	40.0%	75.0%	+35.0 pp
Turns per task	25.2	22.1	−3.1
Shell calls	49.8	46.7	−3.1
Cost per task	$3.26	$2.70	−17%
False positives	0.79	0.33	−0.46
True positives	0.45	0.78	+0.33

The evolved agent reached more accurate conclusions with fewer turns while inspecting a comparable amount of raw evidence. Regressions occurred when upstream search continued after a proximate cause should have ended it.

EnterpriseOps-Gym. Table 4 reports task success and execution traces over the full benchmark. Evolution shortened workflows and increased verifier completion.

Table 4 EnterpriseOps-Gym (GPT-5.4) full-benchmark task-success, execution-trace, and estimated API-cost comparison over 103 tasks.

Metric	Baseline	Evolved	Δ
Full-benchmark task success	23.3%	43.7%	+20.4 pp
Turns per task	18.12	9.87	−8.25
Tool calls	29.53	16.83	−12.70
Cost per task	$1.23	$0.58	−53%
Verifier pass rate	34.5%	72.8%	+38.3 pp

Schema and argument repairs target tool selection. Execution and finish contracts target incomplete workflows. Self-reference and linkage cues preserve state across dependent mutations.

AutomationBench. Table 5 reports the full GPT-5.4 execution-record comparison. On the full GPT-5.4 task set, evolution reduced unsafe execution and improved partial completion.

Table 5 AutomationBench (GPT-5.4) execution-record and estimated API-cost comparison on 100 tasks.

Metric	Baseline	Evolved	Δ
Domain objective score	57.1%	83.2%	+26.1 pp
Mean partial credit	67.3%	86.1%	+18.8 pp
Turns per task	16.35	11.98	−4.37
Cost per task	$0.14	$0.10	−29%
Tasks with guardrail violations	20	4	−16
Total guardrail violations	33	4	−29
Zero-score tasks	24	6	−18

The harness places triage before mutation, anchors dates to the sandbox clock, and delegates arithmetic and spreadsheet operations to deterministic tools. We cannot isolate the contribution of any individual tool from these records.

Across all three benchmarks, the accepted edits repaired interfaces, exposed environment conventions, encoded operational knowledge, or compressed search over available evidence. We cannot isolate the causal contribution of individual patches from the paired comparisons.

6 Conclusion

Our results show evidence that a fixed model can improve substantially when its surrounding harness is adapted to the environment. Across ITBench, EnterpriseOps-Gym, and AutomationBench, StarHarness learned repairs to tool interfaces, environment conventions, operational knowledge, and search-compression aids. These changes improved full-benchmark scores, generalized to tasks excluded from evolution, and transferred across GPT and Qwen model families without re-evolving the harness. Trace analysis associates the gains with shorter workflows, fewer false-positive diagnoses, and fewer unsafe executions in the settings where those measurements were available.

The results suggest that harness evolution is a practical complement to model scaling for stateful enterprise agents. The search changes how a fixed model retrieves records, calls tools, preserves dependencies, and verifies side effects. Future direction is to co-evolve the harness and model weights through reinforcement learning, allowing the scaffold and policy to specialize jointly to enterprise interaction protocols. This could produce smaller, more efficient enterprise models and test whether joint harness–weight optimization can match or exceed larger models at lower inference cost.

References

L. A. Agrawal, S. Tan, D. Soylu, et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. 2025.

Artificial Analysis. Stirrup: A lightweight framework for building agents, 2026. URL https://github.com/ArtificialAnalysis/Stirrup.

Oh My Pi contributors. Oh my pi: A coding-focused variant of the pi agent harness, 2026. URL https://github.com/can1357/oh-my-pi.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, et al. WorkArena: How capable are web agents at solving common knowledge work tasks? In *ICML*, 2024.

P. Hebbar, Y. Manawat, S. Verboomen, et al. SIA: Self improving AI with harness and weight updates. 2026.

S. Jha, R. Arora, Y. Watanabe, et al. ITBench: Evaluating AI agents across diverse real-world IT automation tasks. In *ICML*, 2025.

Andrej Karpathy. Autoresearch: Ai agents running research on single-gpu nanochat training automatically, 2026. URL https://github.com/karpathy/autoresearch.

O. Khattab, A. Singhvi, P. Maheshwari, et al. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *ICLR*, 2024.

Y. Lee, R. Nair, Q. Zhang, et al. Meta-Harness: End-to-end optimization of model harnesses. 2026.

J. Lu, T. Holleis, Y. Zhang, et al. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. 2024.

S. K. R. Malay, S. Nayak, et al. EnterpriseOps-Gym: Environments and evaluations for stateful agentic planning and tool use in enterprise settings. 2026.

Q. Meng, Y. Wang, L. Chen, et al. Agent harness for large language model agents: A survey. 2026. doi: 10.20944/preprints202604.0428.v3.

M. A. Merrill, A. G. Shaw, N. Carlini, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *ICLR*, 2026.

OpenAI. Introducing GPT-5.4, 2026. URL https://openai.com/index/introducing-gpt-5-4/.

K. Opsahl-Ong, M. J. Ryan, J. Purtell, et al. Optimizing instructions and demonstrations for multi-stage language model programs. 2024.

Qwen Team. Qwen3.6: Official model documentation and release repository, 2026. URL https://github.com/QwenLM/Qwen3.6.

B. Rombaut. Inside the scaffold: A source-code taxonomy of coding agent architectures. 2026.

T. Schick, J. Dwivedi-Yu, R. Dessì, et al. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023.

D. Shepard and R. Salimans. AutomationBench. 2026.

Y. Wang, H. Zhu, Z. Hu, et al. Rethinking the evaluation of harness evolution for agents. 2026.

J. Yang et al. SWE-agent: Agent-computer interfaces enable automated software engineering. 2024.

S. Yao, J. Zhao, D. Yu, et al. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

S. Yao, N. Shinn, P. Razavi, and K. Narasimhan. Tau-bench: A benchmark for tool-agent-user interaction in real-world domains. 2024.

Mario Zechner. Pi: An open agent harness and coding agent, 2026. URL https://github.com/earendil-works/pi.

J. Zhang, J. Xiang, Z. Yu, et al. AFlow: Automating agentic workflow generation. In *ICLR*, 2025.