PARETOSADDLER: PARETO-GUIDED OPTIMIZATION OF AGENT HARNESSES FOR QUALITY AND COST

Anonymous authors
Paper under double-blind review

ABSTRACT

Harness optimization can substantially improve LLM agents without modifying model weights, but existing methods primarily optimize response quality. Extending them to joint quality–cost optimization introduces a challenge: applying the deployment preference during intermediate updates can prematurely reject candidates that are valuable for later improvements. We introduce ParetoSaddler, which addresses this mismatch by separating intermediate candidate promotion from final deployment selection. At each iteration, an optimizer updates a parent harness to produce a candidate, and then compares the two over repeated executions and uses Bayesian bootstrap Pareto support to promote candidates showing credible progress in either quality or cost. Promoted candidates receive broader development evaluation and remain available as parents for later optimization steps. We evaluate ParetoSaddler on ClawMark and GAIA2 using domain-disjoint splits. On ClawMark, ParetoSaddler improves quality from 70.86% to 81.35% while reducing cost from $1.17 to $1.01 per task; on GAIA2, it improves quality from 56.22% to 62.89% while reducing cost from $0.78 to $0.45 per task. The Pareto-selected harnesses also achieve higher quality at lower cost than quality-only harness optimization. Analysis shows that Pareto-aware optimization retains intermediate candidates that would be discarded by deployment preference but can support later improvements, while Bayesian bootstrap screening filters out candidates whose gains are weakly supported across runs.

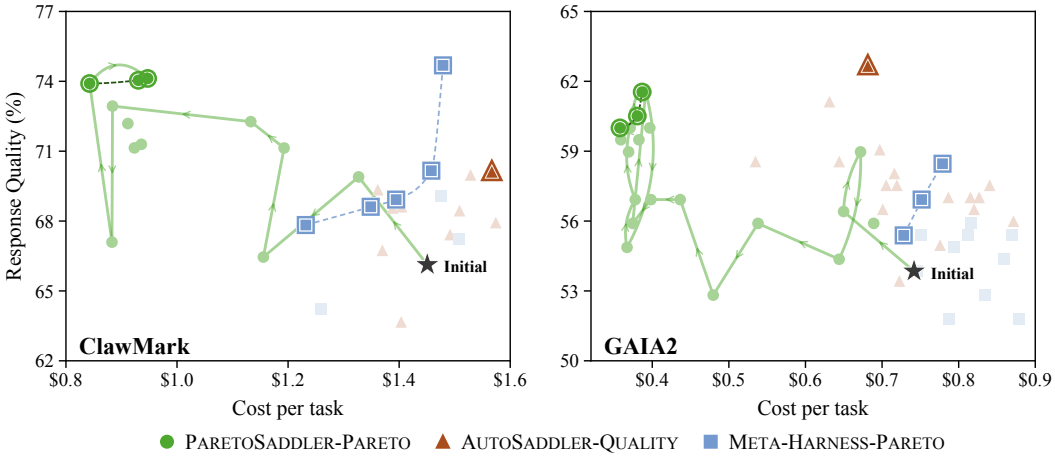

Figure 1: **Quality–cost optimization on the development sets of ClawMark and GAIA2.** Each marker represents a harness evaluated on the development set; better solutions lie toward the upper left. Stars denote the initial harness, and enlarged outlined markers highlight the final candidates retained by each method. Solid green lines trace harness evolution in ParetoSaddler; green markers not connected by solid lines denote candidates not selected for further refinement. Dashed lines connect candidates on the Pareto fronts of ParetoSaddler and Meta-Harness. ParetoSaddler discovers high-quality harnesses in a lower-cost region not reached by the evaluated baseline candidates.

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

1 INTRODUCTION

Large language models (LLMs) have enabled agents that reason, use tools, and interact with executable environments to solve increasingly complex tasks. An agent's capabilities arise from both its underlying model and its *harness*: the prompts, tools, middleware, and control logic that mediate its interaction with the environment. Even with model weights fixed, modifying these components can substantially improve agent performance (LangChain, 2026). This has motivated growing interest in automatic harness optimization (AHO) (Lee et al., 2026; Park et al., 2026), in which an external optimizer diagnoses execution traces and updates the surrounding harness to improve the task agent.

Most existing AHO methods optimize primarily for task quality, whereas real-world deployment must also consider execution cost and other requirements such as safety and privacy. Here, we focus on jointly optimizing task response quality and LLM token cost. Deployment preferences may specify a fixed quality–cost trade-off, impose an explicit constraint, or defer selection among Pareto alternatives. This multi-objective setting raises a key question beyond final deployment selection: *which intermediate harness updates should remain available for continued optimization?*

This distinction is particularly important for AHO methods with *staged evaluation*, where locally generated updates are screened before receiving broader development evaluation (Park et al., 2026). The screening decision does more than reduce evaluation cost: it also determines which intermediate harnesses remain available for subsequent refinement. Applying the final deployment preference directly at this stage can therefore prematurely reject updates whose immediate quality–cost trade-off is unfavorable but whose changes enable stronger solutions later, consistent with prior optimization research showing that search guidance need not coincide with the ultimate objective (Knowles et al., 2001; Lehman & Stanley, 2011). For long-horizon agents, this mismatch can arise naturally because execution cost is often coupled with task progress: a harness may appear inexpensive simply because some executions terminate prematurely, whereas a patch that fixes such failures may improve quality while increasing cost through additional reasoning or environment interaction. Rejecting the update solely because it temporarily violates the deployment cost preference can prevent later patches from building on its capability gains and reducing the newly exposed inefficiencies.

Local promotion is further complicated by stochastic execution. Repeated runs of the same harness can vary substantially in both quality and cost, so an apparent improvement may reflect execution randomness rather than the update itself. Simply comparing mean outcomes collapses this variation into a point estimate and can therefore overstate weakly supported gains. Staged multi-objective AHO thus faces two questions: *what intermediate movements are worth preserving for further optimization, and how can we assess whether their observed gains are statistically supported?*

To address these issues, we introduce **ParetoSaddler**, a framework for staged quality–cost harness optimization that separates *exploration-time promotion* from *deployment-time selection*. At each iteration, an underlying AHO method constructs a candidate harness from execution evidence. ParetoSaddler repeatedly evaluates the candidate and its parent and uses Bayesian bootstrap support (Rubin, 1981) to promote updates showing credible progress in either quality or cost, without requiring them to already satisfy the final deployment preference. Only after the optimization budget is exhausted is that preference applied to select the final harness. In this way, ParetoSaddler separates what is worth *exploring* during optimization from what is ultimately worth *deploying*.

We evaluate ParetoSaddler on ClawMark (Meng et al., 2026) and GAIA2 (Froger et al., 2026) using domain-disjoint training, development, and test splits under both Pareto-set optimization and quality-constrained cost minimization. On ClawMark, ParetoSaddler improves quality on the held-out test set from 70.86% to 81.35% while reducing cost from \$1.17 to \$1.01 per task; on GAIA2, quality improves from 56.22% to 62.89% while cost decreases from \$0.78 to \$0.45. Notably, its quality-oriented Pareto solution also outperforms quality-only harness optimization by 7.49 points on ClawMark and 5.56 points on GAIA2 while remaining cheaper, showing that jointly optimizing quality and cost need not sacrifice quality and can even lead the search to stronger quality solutions.

Our main contributions are summarized as follows:

- When performing multi-objective agent harness optimization with staged evaluation, we identify a mismatch between *intermediate promotion* and *final deployment selection*, where enforcing deployment preferences too early can prune updates useful for later optimization.

- We introduce ParetoSaddler, which combines Pareto-aware promotion, repeated execution, and Bayesian bootstrap screening for effective and reliable quality–cost harness optimization.
- We demonstrate consistent quality–cost improvements on ClawMark and GAIA2, supported by detailed analyses and ablations of promotion, repeated execution, and Bayesian screening.

2 Related Work

Automatic Harness Optimization. Automatic optimization of LLM systems has expanded from prompts (Pryzant et al., 2023; Yuksekgonul et al., 2024; Guo et al., 2024; Yang et al., 2024; Agrawal et al., 2026) and reusable skills (Yang et al., 2026; Ni et al., 2026; Alzubi et al., 2026) to complete agent harnesses. Existing AHO methods broadly follow two optimization protocols. *Search-based* approaches (Lou et al., 2026; Lee et al., 2026; Pan et al., 2026) generate and compare harness candidates using execution feedback, iteratively refining promising candidates. *Learning-based* approaches (Park et al., 2026; Jiang et al., 2026) derive incremental updates from mini-batches and use staged evaluation to decide which updates to retain. These protocols are orthogonal to whether adaptation occurs online during deployment or offline on a fixed development corpus. ParetoSaddler follows the staged offline-learning setting and focuses on joint quality–cost optimization.

Multi-Objective Optimization for LLM Systems. Classical multi-objective optimization retains non-dominated solutions representing different trade-offs (Knowles & Corne, 2000; Zitzler et al., 2001), and recent work applies this principle to prompts (Yang & Li, 2023; Sinha et al., 2024; Büssing et al., 2026) and agent skills (Gong et al., 2026). Full-harness optimization is more costly because evaluating each candidate requires long agent trajectories, motivating staged evaluation that reserves broader evaluation for promising candidates (Schmucker et al., 2021). Prior work shows that search guidance need not coincide with the final selection objective (Knowles et al., 2001; Lehman & Stanley, 2011). Building on this distinction, ParetoSaddler uses Pareto-aware promotion during optimization while reserving the deployment preference for final harness selection.

Optimization under Stochastic Evaluation. Candidate selection under noisy evaluations is a longstanding challenge in optimization (Branke & Schmidt, 2003; Marceau-Caron & Schoenauer, 2014). Recent work has applied repeated evaluation and bootstrap resampling to multi-objective ranking and LLM-agent configuration selection (Rook et al., 2024; Vattikonda et al., 2025). ParetoSaddler builds on this by incorporating Bayesian bootstrap uncertainty estimation (Rubin, 1981) into staged AHO, using statistical support to determine candidate promotion.

3 Preliminaries and Problem Formulation

Agent Harness and Stochastic Execution. An agent harness comprises the prompts, tools, middleware, and control logic surrounding the underlying LLM (Zhou et al., 2026). Let $h \in \mathcal{H}$ denote a candidate harness and m the underlying LLM, which remains fixed throughout optimization. Executing the agent (h, m) on task x induces a stochastic trajectory $\tau \sim p(\tau \mid h, m, x)$, with randomness arising from model generation and interactions with tools and the environment.

Quality and Cost Objectives. We jointly optimize task performance and LLM token cost. Hereafter, we use *quality* as a generic term for benchmark-specific task performance, which may be measured by task success or rubric-based scoring, and *cost* to denote the monetary cost of LLM token usage under a fixed price card. Each trajectory therefore yields a normalized quality score $q(\tau) \in [0, 1]$, with higher values indicating better task performance, and incurs a cost $c(\tau) \geq 0$.

For the j-th call, let $T_{j,\mathrm{in}}$, $T_{j,\mathrm{cache}}$, and $T_{j,\mathrm{out}}$ denote the numbers of fresh-input, cached-input, and output tokens, respectively, with corresponding per-token prices ρ_{in}, ρ_{cache}, and ρ_{out}. For a trajectory τ consisting of $M(\tau)$ model calls, its total cost is then defined as $c(\tau) = \sum_{j=1}^{M(\tau)} (\rho_{\mathrm{in}} T_{j,\mathrm{in}} + \rho_{\mathrm{cache}} T_{j,\mathrm{cache}} + \rho_{\mathrm{out}} T_{j,\mathrm{out}})$. For a target task distribution $\mathcal{P}$, the population-level quality and cost of harness h are

$$Q(h) = \mathbb{E}_{x \sim \mathcal{P}, \tau \sim p(\cdot \mid h, m, x)}[q(\tau)], \qquad C(h) = \mathbb{E}_{x \sim \mathcal{P}, \tau \sim p(\cdot \mid h, m, x)}[c(\tau)]. \tag{1}$$

We seek harnesses with high $Q(h)$ and low $C(h)$. In practice, we estimate these expectations using a finite task dataset $\mathcal{D}$. Given R independent executions per task, let $\tau_{i,r}^h$ denote the r-th trajectory of harness h on task x_i. We average executions within each task and then weight tasks equally:

$$\widehat{Q}_{\mathcal{D}}(h) = \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\frac{1}{R}\sum_{r=1}^{R} q(\tau_{i,r}^h), \qquad \widehat{C}_{\mathcal{D}}(h) = \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\frac{1}{R}\sum_{r=1}^{R} c(\tau_{i,r}^h). \tag{2}$$

Optimization decisions use these empirical estimates on the designated data split.

Deployment Preferences and Optimization Formulations. A deployment preference π specifies which quality–cost trade-off is desirable. Depending on whether these preferences are expressed through fixed weights, explicit constraints, or deferred selection among alternatives, we have:

- **Weighted-Sum Scalarization** combines quality and cost into a single objective:

$$h_{\text{weighted}} \in \arg\max_{h\in\mathcal{H}} \left[Q(h) - \lambda C(h)\right], \tag{3}$$

  where $\lambda > 0$ is a predefined weight balancing quality and cost.

- **Constrained Optimization** optimizes one objective subject to a bound on the other. For example, we formulate *quality-constrained cost minimization* as follows:

$$h_{\text{constrained}} \in \arg\min_{h\in\mathcal{H}} C(h) \quad \text{subject to} \quad Q(h) \geq Q_{\min}, \tag{4}$$

  where $Q_{\min}$ is the quality floor. Alternatively, one can maximize quality under a cost budget.

- **Pareto-Set Optimization** seeks a set of alternatives spanning quality–cost trade-offs when preferences are unspecified. Harness h_a dominates h_b, denoted by $h_a \succ h_b$, if $Q(h_a) \geq Q(h_b)$ and $C(h_a) \leq C(h_b)$, with at least one strict inequality. The Pareto-optimal set is

$$\mathcal{H}^\star = \{h \in \mathcal{H} : \nexists h' \in \mathcal{H} \text{ such that } h' \succ h\}. \tag{5}$$

  In practice, the goal is to approximate this frontier with a diverse set of non-dominated harnesses.

Iterative Harness Optimization with Staged Evaluation. We consider iterative harness optimization in which new candidates build on previously explored harnesses. We use the datasets $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{dev}}$, and $\mathcal{D}_{\text{test}}$ for deriving updates, guiding harness selection, and final assessment, respectively. Because thoroughly evaluating every generated candidate can be expensive, recent prompt and harness optimization and agent self-improvement methods (Agrawal et al., 2026; Park et al., 2026; Zhang et al., 2026) often adopt *staged evaluation*: candidates first undergo a relatively inexpensive screening procedure, and only selected candidates receive broader evaluation. We refer to the decision to advance a candidate beyond this preliminary stage as *promotion*.

4 Proposed Method: ParetoSaddler

Overview. ParetoSaddler brings Pareto reasoning into the promotion decisions of staged harness optimization. As illustrated in Figure 2, at iteration t, the optimizer selects a parent harness h_t and uses generation samples $\mathcal{D}_t^{\text{gen}} \subseteq \mathcal{D}_{\text{train}}$ to construct a candidate h_t' (❶). These samples provide the execution traces and other method-specific evidence used to derive the update. The parent and candidate are then assessed using screening samples $\mathcal{D}_t^{\text{scr}} \subseteq \mathcal{D}_{\text{train}}$, which may differ from or overlap with the generation samples. During screening, the parent and candidate are repeatedly executed under a shared evaluation protocol, yielding joint quality–cost observations.

A statistical Pareto promotion rule (❷) uses these observations to advance a candidate when there is sufficient evidence of improvement in either quality or cost, without requiring the update to satisfy a particular deployment preference. Promoted candidates then receive broader development evaluation (❸), which updates the development-evaluated alternatives and their empirical Pareto archive. The resulting alternatives, together with the deployment preference π, guide continued optimization (❹), forming an iterative loop. After the optimization budget is exhausted, the deployment preference π is applied to the retained candidates to determine the returned harness or harness set (❺). Algorithms 1 and 2 provide the general procedure and its AutoSaddler instantiation, respectively.

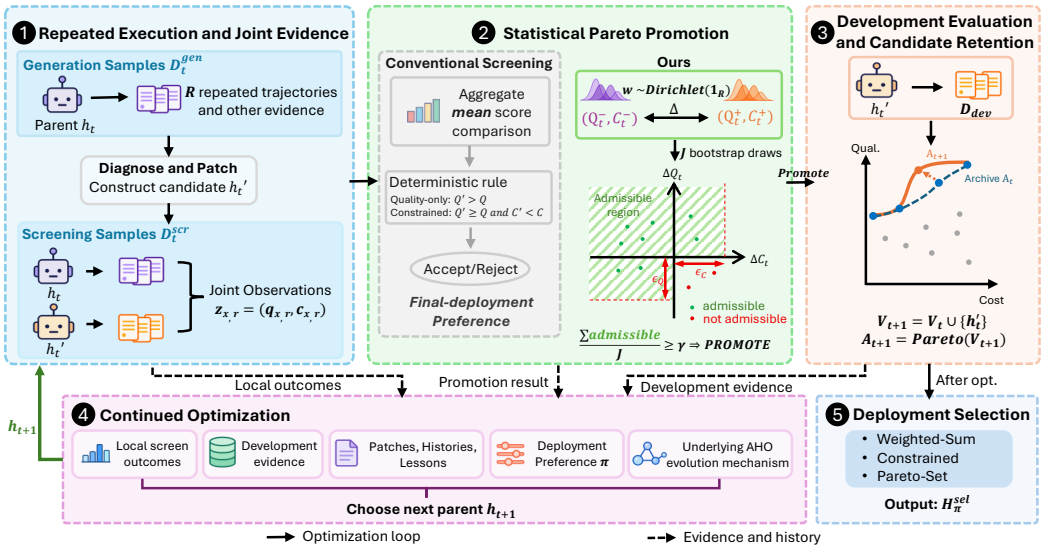

Figure 2: **Overview of ParetoSaddler.** Repeated execution and statistical Pareto promotion identify candidates for development evaluation. The updated candidate pool and Pareto archive support continued optimization, with final harness selection governed by the deployment preference.

Repeated Execution and Evaluation Evidence. Execution randomness affects both quality and cost. As shown in Appendix D, across 20 ClawMark tasks and six repeated runs per task, the mean within-task coefficient of variation reaches **18.8%** for quality and **25.6%** for cost. We therefore use R repeated executions per task for both screening and development evaluation.

Concretely, at iteration t, the underlying AHO method executes the parent harness on the generation context $\mathcal{D}_t^{\text{gen}}$ and collects trajectories together with paired quality–cost observations. For an individual execution τ, we write $\mathbf{z} = (q(\tau), c(\tau))$, while $\mathcal{Z}$ denotes the collection of such observations across executions and tasks. This evidence, together with any method-specific information, is used to construct a candidate h_t'. In our AutoSaddler instantiation, the collected traces are analyzed to diagnose task failures and inefficient behaviors and to propose targeted harness updates.

The candidate is then screened against its parent on $\mathcal{D}_t^{\text{scr}}$, using the same protocol and R executions per task for both harnesses. Let $h_t^- = h_t$ and $h_t^+ = h_t'$, with screening evidence $(\mathcal{T}_t^-, \mathcal{Z}_t^-)$ and $(\mathcal{T}_t^+, \mathcal{Z}_t^+)$, respectively. When generation and screening use compatible tasks and protocols, parent executions can be reused. For example, AutoSaddler generates and screens updates on the same training mini-batch, so $\mathcal{D}_t^{\text{gen}} = \mathcal{D}_t^{\text{scr}}$ and no additional parent executions are required.

Statistical Pareto Promotion. Repeated execution yields multiple quality–cost observations for each task and harness candidate, raising the question of how to consolidate them into a reliable promotion decision. Rather than relying on empirical means, we quantify uncertainty in the observed differences using the Bayesian bootstrap (Rubin, 1981). By assigning continuous Dirichlet weights to the repeated observations, the Bayesian bootstrap induces a nonparametric distribution over plausible quality–cost outcomes, which we use to assess statistical support for candidate improvement.

Specifically, within each bootstrap draw $j = 1, \ldots, J$, we consider each screening sample x and harness $s \in \{-, +\}$ separately. We draw Dirichlet weights over its R observed executions, $(w_{x,1}^{s,(j)}, \ldots, w_{x,R}^{s,(j)}) \sim \text{Dirichlet}(\mathbf{1}_R)$, with $\sum_{r=1}^{R} w_{x,r}^{s,(j)} = 1$, and use the same weight for the quality and cost of each execution. This produces a plausible reweighted outcome for each screening sample. Averaging these outcomes equally across screening samples gives

$$Q_t^{s,(j)} = \frac{1}{|\mathcal{D}_t^{\text{scr}}|} \sum_{x \in \mathcal{D}_t^{\text{scr}}} \sum_{r=1}^{R} w_{x,r}^{s,(j)} q(\tau_{x,r}^{h_t^s}), \qquad C_t^{s,(j)} = \frac{1}{|\mathcal{D}_t^{\text{scr}}|} \sum_{x \in \mathcal{D}_t^{\text{scr}}} \sum_{r=1}^{R} w_{x,r}^{s,(j)} c(\tau_{x,r}^{h_t^s}).$$

We then characterize the candidate's movement in this draw by

$$\Delta Q_t^{(j)} = Q_t^{+,(j)} - Q_t^{-,(j)}, \qquad \Delta C_t^{(j)} = \frac{C_t^{+,(j)} - C_t^{-,(j)}}{C_t^{-,(j)}}, \tag{6}$$

where positive $\Delta Q_t^{(j)}$ denotes a quality gain and negative $\Delta C_t^{(j)}$ denotes a relative cost reduction.

Rather than enforcing a fixed deployment trade-off, we allow improvement in either objective reflecting our key principle: *promote intermediate updates with credible progress along either objective, even if it is not yet desirable for final deployment*. Accordingly, draw j supports promotion when

$$E_t^{(j)} = \left[\Delta Q_t^{(j)} > 0 \wedge \Delta C_t^{(j)} \leq \epsilon_C\right] \vee \left[\Delta C_t^{(j)} < 0 \wedge \Delta Q_t^{(j)} \geq -\epsilon_Q\right]. \tag{7}$$

The allowances $\epsilon_Q, \epsilon_C \geq 0$ serve as loose local safeguards against severe regression in the non-improving objective that may be difficult to recover through subsequent evolution.

Finally, we aggregate the bootstrap outcomes into a promotion-support score:

$$\widehat{p}_t = \frac{1}{J}\sum_{j=1}^{J} \mathbf{1}\left\{E_t^{(j)}\right\}, \qquad \text{promote } h'_t \iff \widehat{p}_t \geq \gamma. \tag{8}$$

Here, $\widehat{p}_t$ is the fraction of bootstrap draws in which the candidate satisfies the promotion event $E_t^{(j)}$. Thus, a candidate is promoted only when it is supported by a γ fraction of the resampled outcomes.

Development Evaluation and Candidate Retention. Promoted candidates are evaluated on $\mathcal{D}_{\text{dev}}$, using R executions per task as well. We maintain the set $\mathcal{V}_t$ of all development-evaluated harnesses and its empirical Pareto archive $\mathcal{A}_t$. After evaluating a promoted candidate,

$$\mathcal{V}_{t+1} = \mathcal{V}_t \cup \{h'_t\}, \qquad \mathcal{A}_{t+1} = \{h \in \mathcal{V}_{t+1} : \nexists g \in \mathcal{V}_{t+1} \text{ such that } g \succ_{\text{dev}} h\}, \tag{9}$$

where $\succ_{\text{dev}}$ denotes dominance under the development estimates. If h'_t is not promoted, both sets remain unchanged. Importantly, $\mathcal{V}_t$ retains all development-evaluated candidates, including those outside the current Pareto archive, preserving intermediate alternatives for subsequent optimization.

Continued Optimization. The development-evaluated set $\mathcal{V}_{t+1}$, Pareto archive $\mathcal{A}_{t+1}$, accumulated optimization state, and deployment preference π then guide construction of the next parent h_{t+1}. Here, π may steer the search toward deployment-relevant regions of the quality–cost space, but does not act as a hard filter: candidates are not excluded from further refinement solely because they violate a deployment constraint or fall outside the current archive.

Deployment Selection. After the optimization budget is exhausted, π is applied definitively to the final development-evaluated set $\mathcal{V}$ according to the weighted, constrained, or Pareto-set formulation in Section 3. This yields $\mathcal{H}_\pi^{\text{sel}}$: a single harness for scalar or constrained selection, or the empirical Pareto set for frontier discovery. Thus, deployment preferences π can guide the search without acting as a hard intermediate filter, while governing the final returned solution.

5 Experiments

5.1 Experimental Setup

Benchmarks, Agents, and Data Splits. We evaluate ParetoSaddler on two challenging long-horizon agent benchmarks: ClawMark (Meng et al., 2026) and GAIA2 (Froger et al., 2026). For both benchmarks, we construct domain-disjoint training, development, and test splits, such that harness updates are evaluated on previously unseen domains. For ClawMark, we use 92 tasks with stage-0 rubrics and optimize OpenClaw (OpenClaw, 2026) with GPT-5.4 at high reasoning effort. For GAIA2, we follow the same split configuration as AutoSaddler (Park et al., 2026) and optimize the benchmark's default ReAct-style harness using GPT-5.5 with medium reasoning effort.

Table 1: Test results on ClawMark and GAIA2. Quality is reported in percentage (%), and cost in USD per task. Values are mean ± sample standard deviation across three test repeats. **Bold** and underlined means indicate the best and second-best results in each column, respectively. GAIA2 universe-level results are also shown as a detailed breakdown. For Pareto-set optimization, superscripts Q and C denote the quality- and cost-extreme harnesses selected on the development set.

Setting	Metric	Overall		GAIA2 by Universe		
		ClawMark	**GAIA2**	U21 (107)	U22 (112)	U27 (81)
Default Agent	Qual. ↑	70.86 ± 2.29	56.22 ± 1.07	58.26 ± 1.08	53.57 ± 1.55	57.20 ± 2.57
	Cost ↓	1.17 ± 0.17	0.78 ± 0.08	0.82 ± 0.18	0.71 ± 0.03	0.82 ± 0.04
AS-QUALITY	Qual. ↑	73.86 ± 3.54	57.33 ± 0.67	55.76 ± 4.70	53.87 ± 5.67	64.20 ± 1.23
	Cost ↓	1.24 ± 0.04	0.61 ± 0.03	0.55 ± 0.02	0.62 ± 0.04	0.67 ± 0.05
MH-MINCOST	Qual. ↑	70.02 ± 2.35	39.78 ± 0.96	41.12 ± 2.80	38.69 ± 2.58	39.51 ± 2.47
	Cost ↓	1.16 ± 0.05	0.62 ± 0.03	0.64 ± 0.04	0.57 ± 0.01	0.65 ± 0.02
MH-PARETO^C	Qual. ↑	70.50 ± 3.92	55.44 ± 1.50	58.26 ± 3.28	50.00 ± 2.68	59.26 ± 3.27
	Cost ↓	0.97 ± 0.05	0.80 ± 0.01	0.76 ± 0.01	0.78 ± 0.02	0.89 ± 0.02
MH-PARETO^Q	Qual. ↑	73.25 ± 5.88	54.33 ± 2.03	57.63 ± 3.28	49.40 ± 1.36	56.79 ± 3.27
	Cost ↓	1.06 ± 0.06	0.83 ± 0.04	0.87 ± 0.06	0.75 ± 0.01	0.89 ± 0.07
PS-MINCOST	Qual. ↑	75.16 ± 2.44	56.11 ± 1.71	58.26 ± 0.54	50.30 ± 2.25	61.32 ± 3.77
	Cost ↓	**0.82** ± 0.07	**0.44** ± 0.01	**0.37** ± 0.02	**0.48** ± 0.01	0.48 ± 0.01
PS-PARETO^C	Qual. ↑	79.60 ± 2.43	62.11 ± 2.46	**63.24** ± 4.80	59.23 ± 1.03	64.61 ± 3.77
	Cost ↓	0.92 ± 0.08	0.47 ± 0.00	0.44 ± 0.01	0.49 ± 0.02	0.47 ± 0.00
PS-PARETO^Q	Qual. ↑	**81.35** ± 3.11	**62.89** ± 2.14	61.99 ± 2.35	**60.71** ± 2.36	**67.08** ± 1.89
	Cost ↓	1.01 ± 0.02	0.45 ± 0.02	0.39 ± 0.01	0.49 ± 0.03	**0.46** ± 0.02

Instantiation and Baselines. We instantiate ParetoSaddler on top of AutoSaddler. For deployment preferences, we study two representative settings: **(i) Pareto-set optimization**, where no specific quality–cost preference is specified in advance, and **(ii) quality-constrained cost minimization**, where cost is minimized subject to a required quality level.

We compare ParetoSaddler against two baselines. **(i) Meta-Harness** (Lee et al., 2026) is a representative harness-optimization method without staged evaluation, iteratively updating the harness using the full training set. We adapt MH to the two deployment preferences considered above. **(ii) AutoSaddler** (Park et al., 2026) serves as our single-objective, quality-oriented baseline, allowing us to assess whether jointly optimizing quality and cost comes at the expense of quality improvement. For conciseness, we denote the evaluated method–preference combinations as PS-PARETO and PS-MINCOST for ParetoSaddler under Pareto-set optimization and quality-constrained cost minimization, respectively; MH-PARETO and MH-MINCOST for the corresponding Meta-Harness variants; and AS-QUALITY for our quality-only AutoSaddler adaptation. For Pareto-set optimization, we select two representative harnesses from the development Pareto set for test evaluation: the quality-extreme harness (e.g., PS-PARETO^Q) and the cost-extreme harness (e.g., PS-PARETO^C).

Experimental Protocol. We use the GitHub Copilot SDK (GitHub, 2026) with GPT-5.5 at xhigh reasoning effort as the optimizer for all methods (usage reported in Appendix G). Unless otherwise stated, each task is evaluated with $R = 3$ rollouts, and ParetoSaddler and AutoSaddler use training mini-batches of three tasks. For Bayesian screening, we use $J = 10{,}000$ bootstrap draws, a prespecified promotion threshold of $\gamma = 0.7$, and a local quality-loss allowance of $\epsilon_Q = 0.1$. We impose no upper bound on local cost increases, setting $\epsilon_C = +\infty$. Moreover, for PS-MINCOST we require the quality to remain within 10 percentage points below that of the initial harness. We measure response quality using normalized stage-0 rubric score on ClawMark, and Pass@1 on GAIA2, and report token-based execution cost in USD per task. Full experimental details are provided in Appendix A.

5.2 MAIN RESULTS

Table 1 reports the performance of development-selected harnesses on the domain-disjoint test sets, while Figure 1 shows their evolution in quality–cost space on the development set.

ParetoSaddler improves both quality and cost on unseen domains. The quality-extreme PS-PARETO^Q achieves the highest mean test quality on both benchmarks while reducing cost relative to

the initial harness. On ClawMark, it improves quality by **14.8%** (70.86% to 81.35%) while reducing cost by **13.7%** ($1.17 to $1.01); on GAIA2, it improves quality by **11.9%** (56.22% to 62.89%) while reducing cost by **42.3%** ($0.78 to $0.45 per task). The cost-extreme PS-PARETO^C also improves quality and reduces cost on both benchmarks. Both operating points outperform their corresponding Meta-Harness variants in mean quality and cost on both benchmarks. These gains are consistent across all three held-out GAIA2 universes (see Appendix F for additional stability analysis).

Joint optimization does not sacrifice quality, and may even improve it. Despite optimizing both quality and cost, PS-PARETO^Q outperforms the quality-only AS-QUALITY baseline by 7.49 quality points on ClawMark and 5.56 points on GAIA2, while also reducing cost on both benchmarks. This is a somewhat unexpected result: introducing the cost objective is associated with better, rather than merely preserved, quality. We examine this effect and the role of the local Pareto gate in RQ1.

Deployment preference steers optimization toward different operating regions. Under quality-constrained cost minimization, PS-MINCOST moves further toward lower-cost solutions while maintaining the required quality level. It reaches $0.82 per task on ClawMark and $0.44 on GAIA2, lower than both development-selected PS-PARETO operating points. It also dominates MH-MINCOST on both benchmarks in both quality and cost.

Pareto reasoning preserves useful quality–cost alternatives. As shown in Figure 1, PS-PARETO explores and retains multiple operating points along the development quality–cost frontier rather than committing the search to a single trade-off. This enables distinct quality- and cost-oriented harnesses to be selected after optimization according to the deployment preference.

5.3 ANALYSIS AND ABLATION STUDIES

RQ1: How does Pareto-aware promotion enable better optimization outcomes? We examine the promotion decisions on ClawMark, with complementary GAIA2 cases in Appendix C.

Pareto-set vs. quality-only optimization. I-7 in PS-PARETO provides a representative example (Figure 7). Its parent achieves perfect quality on all three mini-batch tasks across three executions. I-7 reduces mean mini-batch cost by 17.22% with a 1.33-point quality decrease, which the quality gate would reject. Pareto-aware promotion instead advances I-7 to development evaluation, where it improves both quality (72.94% vs. 72.27%) and cost ($0.88 vs. $1.13). Its bounded-probing and completion-control behaviors persist in later iterations, contributing to the selected harness.

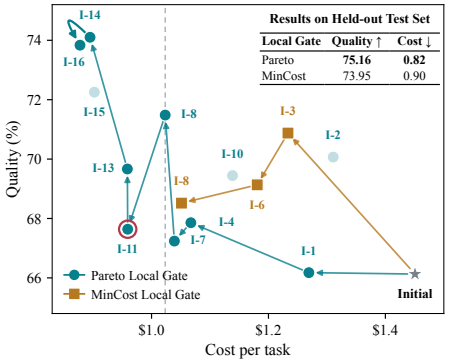

Figure 3: ClawMark optimization trajectories and final test results of PS-MINCOST under Pareto and MinCost local gates. Faint markers denote evaluated candidates not selected for subsequent refinement.

Controlled gate ablation. Because the preceding analysis is observational, we next perform a controlled study by replacing Pareto-aware promotion with a deployment-oriented *MinCost Local Gate* in the PS-MINCOST setting. Figure 3 shows that this change produces markedly different search trajectories. The Pareto gate promotes 11 candidates, compared with only three under the MinCost Local Gate, and reaches a better test operating point. The key divergence occurs around I-11, before which the Pareto trajectory approaches a similar cost "wall" to the MinCost Local Gate, near the dashed line in Figure 3. I-11 improves mini-batch quality from 83.65% to 87.62% while increasing cost by 10.5%. The MinCost gate would reject this update, whereas Pareto promotion retains it for development evaluation and subsequent refinement toward I-14 and I-16. Without admitting this intermediate update, the Pareto trajectory would likely remain trapped near the same local cost minima as the MinCost Local Gate.

Together, these results show that Pareto-aware promotion enables better optimization by preserving intermediate states whose local trade-offs may appear suboptimal, but whose value emerges through broader evaluation and subsequent refinement.

RQ2: How do repeated executions improve harness optimization? We ask what additional evidence repeated executions provide and whether this evidence improves optimization outcomes.

Diagnostic value of repeated executions. Repeated runs reveal three types of evidence that a single trajectory cannot provide: *(i) persistent failures*, where repeated failure across runs indicates recurring defects; *(ii) mixed outcomes*, where quality variation on the same task provides direct contrast for unstable behaviors; and *(iii) efficiency contrast*, where all runs succeed but differ substantially in execution cost. Table 7 provides more details. Repeated executions also strengthen *patch verification and promotion*: pre- and post-patch comparisons reveal whether an improvement persists, while repeated observations provide more reliable evidence for promotion than a single atypical execution.

Single-run vs. three-run optimization. As the analysis above is again observational, we perform a controlled end-to-end ablation within the PS-PARETO setting, varying only the number of executions per task from one to three while keeping the remaining optimization setup fixed. Table 2 shows that three-run optimization reaches stronger held-out operating points. At the cost-oriented endpoint, it achieves 79.60% quality at $0.92 per task, compared with 76.95% at $0.95 for single-run optimization. At the quality-oriented endpoint, held-out quality increases from 75.71% to 81.35%.

Table 2: Overall ClawMark test results for the single- versus three-run PS-PARETO ablation.

Setting	Qual. ↑	Cost ↓
Single-run^C	76.95 ± 5.50	0.95 ± 0.06
Single-run^Q	75.71 ± 2.10	0.97 ± 0.05
Three-run^C	79.60 ± 2.43	$\mathbf{0.92} \pm 0.08$
Three-run^Q	$\mathbf{81.35} \pm 3.11$	1.01 ± 0.02

RQ3: Does Bayesian screening improve promotion under execution uncertainty? While repeated executions provide richer optimization evidence, this evidence must still be consolidated for promotion. A straightforward mean-based gate compares average quality–cost movements, whereas our Bayesian gate also accounts for the statistical support behind that movement.

Retrospective analysis of promotion decisions. We replay mean-based screening on the same 17 candidates generated by PS-PARETO on ClawMark. The mean-based gate would promote 14 candidates, versus 12 under Bayesian screening. I-12 illustrates how aggregation can obscure the underlying quality–cost movement. Its mean mini-batch quality drops by 5.55 points while cost decreases by 7.19%, satisfying the configured quality-loss allowance. Yet Table 9 shows that this aggregate trade-off is highly uneven: nearly all of the cost reduction comes from *Research assistant 4*, where quality also drops sharply, whereas the other two tasks become more expensive. Retrospective development evaluation further shows that I-12 is 2.80 quality points worse and 10.66% more expensive than its parent.

Table 3: ClawMark test results with mean-based versus Bayesian screening.

Setting	Metric	Mean	Bayesian
PS-PARETO^C	Qual. ↑	76.86 ± 5.41	$\mathbf{79.60} \pm 2.43$
	Cost ↓	1.09 ± 0.09	$\mathbf{0.92} \pm 0.08$
PS-PARETO^Q	Qual. ↑	76.53 ± 2.12	$\mathbf{81.35} \pm 3.11$
	Cost ↓	1.15 ± 0.04	$\mathbf{1.01} \pm 0.02$

Controlled ablation: Bayesian vs. mean-based screening. Since the retrospective analysis cannot capture how promotion decisions alter subsequent search, we further perform a full ablation within PS-PARETO, replacing Bayesian screening with the mean-based gate while keeping the remaining setup fixed. As shown in Table 3, Bayesian screening reaches stronger test operating points: its cost-oriented solution achieves 79.60% quality at $0.92 per task, versus 76.86% at $1.09 for mean-based screening, while its quality-oriented solution reaches 81.35% at $1.01, versus 76.53% at $1.15.

Sensitivity to the promotion threshold. Sensitivity analysis in Appendix E shows that $\gamma = 0.7$ lies in a locally stable region on both benchmarks, supporting its use as a default. For other domains, a small-scale grid search on γ may be useful if additional compute is available.

6 CONCLUSION

We introduced ParetoSaddler, a framework for staged quality–cost harness optimization that separates intermediate candidate promotion from final deployment selection. On ClawMark and GAIA2, its Pareto-selected harnesses achieve higher quality at lower execution cost than quality-only harness optimization. Analyses and ablations show that Pareto-aware promotion preserves intermediate candidates that can enable later improvements, repeated executions provide richer and more reliable evidence for optimization, and Bayesian screening filters candidates whose apparent quality–cost gains are weakly supported under execution uncertainty. These findings highlight the value of preserving intermediate trade-offs and accounting for execution uncertainty in harness optimization.

7 AI USE STATEMENT

We used generative AI tools to assist with writing and polishing manuscript sections. We have not used generative AI tools for research ideation, methodology, or experimental design. All research concepts and analyses were developed and conducted by the authors. The authors reviewed the AI-assisted text, checked numerical claims against experimental records, and verified cited sources. We take full responsibility for the final manuscript, including its claims, results, and conclusions.

8 REPRODUCIBILITY STATEMENT

We have made every effort to support the reproducibility of the results presented in this paper. Source code is available through the anonymous repository. Section 4 and Appendix B describe the method and provide pseudocode for the general framework and its AutoSaddler instantiation. Section 5.1 and Appendix A document the benchmark splits, model configurations, adaptations, optimization schedules, and other experimental setup details. Appendix H presents annotated optimization trajectories, while Appendices I and J document complete runs. Appendix K provides selected optimizer prompt excerpts. Both benchmarks, ClawMark and GAIA2, are publicly available.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026.

Jürgen Branke and Christian Schmidt. Selection in the presence of noise. In Erick Cantú-Paz, James A. Foster, Kalyanmoy Deb, Lawrence David Davis, Rajkumar Roy, Una-May O'Reilly, Hans-Georg Beyer, Russell Standish, Graham Kendall, Stewart Wilson, Mark Harman, Joachim Wegener, Dipankar Dasgupta, Mitch A. Potter, Alan C. Schultz, Kathryn A. Dowsland, Natasha Jonoska, and Julian Miller (eds.), *Genetic and Evolutionary Computation — GECCO 2003*, pp. 766–777, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45105-1.

Jan Büssing, Moritz Schlager, Timo Heiß, Tom Zehle, and Matthias Feurer. Mo-capo: Multi-objective cost-aware prompt optimization. *arXiv preprint arXiv:2605.18869*, 2026.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, et al. Gaia2: Benchmarking llm agents on dynamic and asynchronous environments. In *International Conference on Learning Representations*, volume 2026, pp. 119758–119789, 2026.

GitHub. Github copilot sdk. `https://docs.github.com/en/copilot/how-tos/copilot-sdk`, 2026. Accessed: 2026-09-08.

Jingzhi Gong, Ruizhen Gu, Zhiwei Fei, Yazhuo Cao, Lukas Twist, Alina Geiger, Shuo Han, Dominik Sobania, Federica Sarro, and Jie M Zhang. Skillmoo: Multi-objective optimization of agent skills for software engineering. *arXiv preprint arXiv:2604.09297*, 2026.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 34133–34156, 2024.

Wen Jiang, Mingmin Chu, Yimeng Tian, Qianxin Zhang, Haofei Yang, Rui Yang, Yang Liu, Tao Lv, and Fangming Li. Harnessevolve: Learning from reference trajectories for reliable agent self-evolution. *arXiv preprint arXiv:2609.00829*, 2026.

Joshua D Knowles and David W Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary computation*, 8(2):149–172, 2000.

Joshua D. Knowles, Richard A. Watson, and David W. Corne. Reducing local optima in single-objective problems by multi-objectivization. In Eckart Zitzler, Lothar Thiele, Kalyanmoy Deb, Carlos Artemio Coello Coello, and David Corne (eds.), *Evolutionary Multi-Criterion Optimization*, pp. 269–283, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-44719-1.

LangChain. Improving deep agents with harness engineering. https://www.langchain.com/blog/improving-deep-agents-with-harness-engineering, 2026. Accessed 2026-04-21.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 06 2011. ISSN 1063-6560. doi: 10.1162/EVCO_a_00025. URL https://doi.org/10.1162/EVCO_a_00025.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P Murphy. Autoharness: improving llm agents by automatically synthesizing a code harness. *arXiv preprint arXiv:2603.03329*, 2026.

Gaetan Marceau-Caron and Marc Schoenauer. Racing multi-objective selection probabilities. In Thomas Bartz-Beielstein, Jürgen Branke, Bogdan Filipič, and Jim Smith (eds.), *Parallel Problem Solving from Nature – PPSN XIII*, pp. 631–640, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10762-2.

Fanqing Meng, Lingxiao Du, Zijian Wu, Guanzheng Chen, Xiangyan Liu, Jiaqi Liao, Chonghe Jiang, Zhenglin Wan, Jiawei Gu, Pengfei Zhou, et al. Clawmark: A living-world benchmark for multi-turn, multi-day, multimodal coworker agents. *arXiv preprint arXiv:2604.23781*, 2026.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

OpenClaw. Openclaw: Your own personal ai assistant. https://github.com/openclaw/openclaw, 2026. GitHub repository. Accessed: 2026-08-25.

Wenbo Pan, Shujie Liu, Chin-Yew Lin, Jingying Zeng, Xianfeng Tang, Xiangyang Zhou, Yan Lu, and Xiaohua Jia. Evolving agents in the dark: Retrospective harness optimization via self-preference. *arXiv preprint arXiv:2606.05922*, 2026.

Sungho Park, Wonjoong Kim, Rongyuan Tan, Jue Zhang, Wook-Shin Han, Pengfei Gao, Chanyoung Park, Yongqiang Yao, Rao Fu, Elsie Nallipogu, et al. Autosaddler: Automatic harness optimization with durable updates from agent execution traces. *arXiv preprint arXiv:2608.23041*, 2026.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 7957–7968, 2023.

Jeroen Rook, Holger H. Hoos, and Heike Trautmann. Multi-objective ranking using bootstrap resampling. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '24 Companion, pp. 155–158, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704956. doi: 10.1145/3638530.3654436. URL https://doi.org/10.1145/3638530.3654436.

Donald B Rubin. The bayesian bootstrap. *The annals of statistics*, pp. 130–134, 1981.

Robin Schmucker, Michele Donini, Muhammad Bilal Zafar, David Salinas, and Cédric Archambeau. Multi-objective asynchronous successive halving, 2021. URL https://arxiv.org/abs/2106.12639.

Ankita Sinha, Wendi Cui, Kamalika Das, and Jiaxin Zhang. Survival of the safest: Towards secure prompt optimization through interleaved multi-objective evolution. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 1016–1027, 2024.

Dheeraj Vattikonda, Santhoshi Ravichandran, Emiliano Penaloza, Hadi Nekoei, Thibault de Chezelles, Megh Thakkar, Nicolas Gontier, Miguel Muñoz Mármol, Sahar Omidi Shayegan, Stefania Raimondo, Steve (Xue) Liu, Alexandre Drouin, Alexandre Piche, Alexandre Lacoste, and Massimo Caccia. How to train your llm web agent: A statistical diagnosis. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 53256–53282. Curran Associates, Inc., 2025. doi: 10.52202/085713-1774. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/4d239452d41ded07139801d7c966c629-Paper-Conference.pdf.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 12028–12068, 2024.

Heng Yang and Ke Li. Instoptima: Evolutionary multi-objective instruction optimization via large language model-based instruction operators. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 13593–13602, 2023.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, et al. Skillopt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents, 2026. URL https://arxiv.org/abs/2603.19461.

Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, et al. Externalization in llm agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*, 2026.

Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. *TIK report*, 103, 2001.

A Additional Experimental Details

Benchmarks and Data Splits. We evaluate ParetoSaddler on two long-horizon agent benchmarks with complementary execution settings. ClawMark (Meng et al., 2026) evaluates multi-turn, multi-day coworker agents across stateful professional scenarios, covering realistic workflows that involve persistent state, tool use, and rule-based evaluation. GAIA2 (Froger et al., 2026) evaluates agents in dynamic and asynchronous environments, where the environment may evolve independently of agent actions and tasks require sustained interaction with external services.

We construct training, development, and test splits at the group level rather than randomly partitioning individual tasks. This prevents closely related tasks from the same group from appearing in both optimization and evaluation, and tests whether harness updates learned during optimization transfer to previously unseen task groups. The same splits are fixed before optimization and shared by all methods. For ClawMark, we restrict evaluation to initial-stage tasks with explicit stage-0 rubrics to keep repeated evaluations across optimization candidates within the available execution budget. Later-stage adaptation is outside the scope of this evaluation. For GAIA2, we follow the split protocol of AutoSaddler (Park et al., 2026). The resulting assignments are summarized in Table 4.

Table 4: Group-disjoint benchmark splits. ClawMark is partitioned by scenario and GAIA2 by Universe. Numbers in parentheses denote the number of tasks.

Benchmark	Train	Dev	Test
ClawMark	Clinical Assistant, Content Operation, EDA, Executive Assistant, HR, Research Assistant (50)	Investment Analyst, Project Management, Real Estate (20)	E-commerce, Journalist, Legal Assistant (22)
GAIA2	Universe 29 (75)	Universe 30 (65)	Universes 21, 22, 27 (300)

Cost Accounting. We use GPT-5.4 with high reasoning effort for ClawMark experiments, and GPT-5.5 with medium reasoning effort for GAIA2 experiments. Table 5 reports the token prices used to compute execution cost according to $c(\tau) = \sum_{j=1}^{M(\tau)} (\rho_{\text{in}} T_{j,\text{in}} + \rho_{\text{cache}} T_{j,\text{cache}} + \rho_{\text{out}} T_{j,\text{out}})$. We charge fresh-input, cached-input, and output tokens separately using the price card of the corresponding task model.

Table 5: Task-model configurations and token prices used for execution-cost accounting. Prices are in USD per one million tokens.

Benchmark	Model	Reasoning	Fresh input	Cached input	Output
ClawMark	GPT-5.4	High	\$2.50	\$0.25	\$15.00
GAIA2	GPT-5.5	Medium	\$5.00	\$0.50	\$30.00

We normalize execution cost to a cold-start accounting convention: cached input tokens reported for the first valid model call in each task execution are charged at the fresh-input rate. Subsequent calls use the reported fresh-input and cache-read token counts at their respective rates. This convention prevents pre-existing cache state from discounting the initial call in the reported execution cost.

Deployment Settings. We evaluate two settings for deployment preference: Pareto-set optimization and quality-constrained cost minimization. The symmetric setting of cost-constrained quality maximization can be handled analogously. We do not separately evaluate weighted-sum scalarization, as its fixed scalar objective can be optimized by existing single-objective AHO methods.

Benchmark Adaptation. All ClawMark experiments use OpenClaw v2026.6.9. We adapt ClawMark to reduce cross-run interference during parallel evaluation. Each execution receives a unique Notion root, replacing shared evaluation-hub cleanup with execution-scoped teardown. For Google Sheets, we retain per-execution Drive folders and strengthen exact-ID resource lookup. Execution-specific resource IDs are injected into each stage's agent context and reused in checker-side resolution, while tool-use guidance discourages global title-based discovery. These changes preserve task objectives and rubric criteria.

Baseline Adaptation. We adapt AutoSaddler and Meta-Harness to the same benchmark environments and monetary-cost accounting used by ParetoSaddler, while preserving their distinct search and selection mechanisms. Because the methods differ in their evaluation structure, we report their evaluation schedules and repetition counts explicitly rather than forcing an identical protocol.

For AutoSaddler (Park et al., 2026), we use its quality-only configuration, denoted AS-QUALITY. We combine its separate capability- and steering-oriented optimization phases into a unified search loop, allowing the optimizer to apply either or both types of changes within each iteration. This aligns the available edit choices with ParetoSaddler and avoids confounding the comparison of optimization objectives with a fixed phase schedule. Each task is executed three times, with quality averaged across executions before aggregation across tasks. We also apply a comparable Bayesian-bootstrap test when deciding whether to promote a local candidate based on quality alone, using the same promotion threshold γ. Proposal generation and candidate selection therefore remain quality-oriented; execution cost is only recorded for evaluation but is not used as an optimization objective or promotion criterion.

For Meta-Harness (Lee et al., 2026), we retain its history-conditioned proposal and evaluation loop without introducing staged evaluation. We instantiate two objective-specific variants. MH-PARETO receives both quality and monetary-cost feedback, maintains the nondominated quality–cost frontier, and draws proposal parents from this frontier. MH-MINCOST instead minimizes mean execution cost subject to the baseline-relative quality constraint

$$Q(h) \geq \max\{0, Q(h_0) - \epsilon\},$$

using the same absolute tolerance ϵ as PS-MINCOST. The lowest-cost feasible candidate serves as the selection reference, while other historical candidates still contribute to subsequent proposals.

Optimization Details. Unless otherwise stated, we use $R = 3$ executions per task. This choice reflects the substantial cost of repeatedly executing long-horizon agents across optimization candidates and the available evaluation budget. With only three executions, rare outcomes may remain unobserved, and estimates of execution variability and bootstrap promotion support may be sensitive to the observed runs, particularly for candidates near the promotion threshold. Larger repetition counts could improve statistical precision but would require additional evaluation resources. We use $J = 10{,}000$ independent Bayesian bootstrap draws, limiting the Monte Carlo standard error of the estimated promotion support to at most 0.005. These draws reuse the recorded observations and require no additional LLM calls. The cost safeguard is disabled by setting $\epsilon_C = +\infty$. This is because a substantial local cost increase can still reflect genuine task progress. For example, resolving premature termination may result in substantial task progress while requiring more reasoning or environment interactions, making the cost increase difficult to estimate beforehand.

ParetoSaddler and AS-QUALITY perform one optimization epoch over the training split using mini-batches of three tasks. This yields 17 proposal iterations on ClawMark and 25 on GAIA2. Unless otherwise stated, each task is executed three times. The task model and its inference configuration remain fixed throughout optimization.

Meta-Harness uses its original full-set evaluation structure rather than staged evaluation. Each proposed harness is evaluated on the combined training and development search set. We run Meta-Harness for 8 proposal iterations on ClawMark and 15 on GAIA2, giving Meta-Harness approximately $1.37\times$ and $1.22\times$ the maximum task-execution budgets of the staged methods on ClawMark and GAIA2, respectively. We allocate Meta-Harness a larger evaluation budget to avoid favoring the methods using staged evaluation through additional optimization compute.

B ALGORITHM OF PARETOSADDLER

General Algorithm. Algorithm 1 gives the complete procedure for iterative AHO with staged evaluation. We use the same notation as Section 4. At iteration t, generation samples $\mathcal{D}_t^{\text{gen}} \subseteq \mathcal{D}_{\text{train}}$ provide the evidence used to construct h'_t, while screening samples $\mathcal{D}_t^{\text{scr}} \subseteq \mathcal{D}_{\text{train}}$ provide the repeated parent–candidate evidence used for promotion. The two sample sets may differ or overlap, and compatible parent executions may be reused.

Let $h_t^- = h_t$ and $h_t^+ = h'_t$. For $s \in \{-, +\}$, repeated screening produces

$$\mathcal{T}_t^s = \{\tau_{x,r}^{h_t^s} : x \in \mathcal{D}_t^{\text{scr}},\ r = 1, \ldots, R\}, \qquad \mathcal{Z}_t^s = \{\mathbf{z}_{x,r}^{h_t^s} : x \in \mathcal{D}_t^{\text{scr}},\ r = 1, \ldots, R\},$$

Table 6: **Optimization schedules and task-execution budgets.** A task execution denotes one complete rollout of the task agent on one benchmark task. For ParetoSaddler (PS) and AutoSaddler (AS), the reported budget is an upper bound that assumes every locally screened candidate is promoted to full-development evaluation; the actual number of task executions is therefore less than or equal to this value and depends on the number of local promotions. Meta-Harness (MH) does not use staged promotion and instead evaluates each proposal on the full search set, so its budget is determined directly by the number of proposals shown. All budgets include the initial default harness evaluation.

Benchmark	Method	Proposals	Search tasks / proposal	Repeats	Task-execution budget
ClawMark	PS / AS	17	3 + promoted dev. eval.	3	1380
	MH	8	70	3	1890
GAIA2	PS / AS	25	3 + promoted dev. eval.	3	5520
	MH	15	140	3	6720

where

$$\mathbf{z}_{x,r}^{h_t^s} = \big(q(\tau_{x,r}^{h_t^s}), c(\tau_{x,r}^{h_t^s})\big).$$

Bootstrap Pareto Support. For completeness, we give the exact estimator used by BOOTSTRAPPARETOSUPPORT. For each bootstrap draw $j = 1, \ldots, J$, screening sample $x \in \mathcal{D}_t^{\mathrm{scr}}$, and $s \in \{-, +\}$, independently draw

$$\mathbf{w}_{t,x}^{s,(j)} \sim \mathrm{Dirichlet}(\mathbf{1}_R), \qquad \sum_{r=1}^{R} w_{t,x,r}^{s,(j)} = 1.$$

Applying each weight jointly to the corresponding quality and cost gives the task-balanced estimates

$$Q_t^{s,(j)} = \frac{1}{|\mathcal{D}_t^{\mathrm{scr}}|} \sum_{x \in \mathcal{D}_t^{\mathrm{scr}}} \sum_{r=1}^{R} w_{t,x,r}^{s,(j)} q(\tau_{x,r}^{h_t^s}), \qquad C_t^{s,(j)} = \frac{1}{|\mathcal{D}_t^{\mathrm{scr}}|} \sum_{x \in \mathcal{D}_t^{\mathrm{scr}}} \sum_{r=1}^{R} w_{t,x,r}^{s,(j)} c(\tau_{x,r}^{h_t^s}).$$

We then compute

$$\Delta Q_t^{(j)} = Q_t^{+,(j)} - Q_t^{-,(j)}, \qquad \Delta C_t^{(j)} = \frac{C_t^{+,(j)} - C_t^{-,(j)}}{C_t^{-,(j)}},$$

and define

$$E_t^{(j)} = \big[\Delta Q_t^{(j)} > 0 \wedge \Delta C_t^{(j)} \le \epsilon_C\big] \vee \big[\Delta C_t^{(j)} < 0 \wedge \Delta Q_t^{(j)} \ge -\epsilon_Q\big].$$

The estimated promotion support is

$$\widehat{p}_t = \frac{1}{J} \sum_{j=1}^{J} \mathbf{1}\Big\{E_t^{(j)}\Big\}.$$

Algorithm 1 promotes the candidate when $\widehat{p}_t \ge \gamma$.

AutoSaddler Instantiation. Algorithm 2 instantiates ParetoSaddler on AutoSaddler (Park et al., 2026), with adaptations highlighted in orange. At iteration t, the same training mini-batch is used for generation and screening,

$$\mathcal{D}_t^{\mathrm{gen}} = \mathcal{D}_t^{\mathrm{scr}} = \mathcal{B}_t,$$

so the repeated parent executions are reused as generation evidence. AutoSaddler diagnoses task failures and inefficient behaviors from these executions, proposes a targeted patch, and evaluates the resulting candidate on the same mini-batch before applying the Pareto promotion gate. Promoted candidates receive development evaluation; subsequent reflection and EvoDAG evolution use the accumulated evidence together with $\mathcal{V}_{t+1}$, $\mathcal{A}_{t+1}$, and π.

Algorithm 1 ParetoSaddler for iterative AHO with staged evaluation

Require: Initial harness h_0, development set $\mathcal{D}_{\text{dev}}$, optimization budget K, number of evaluation repeats R, number of bootstrap draws J, parent-relative local allowances ϵ_Q, ϵ_C, promotion threshold γ, and deployment preference π

Evaluate h_0 on $\mathcal{D}_{\text{dev}}$ with R repeats and record the results
Initialize optimization state $\mathcal{S}_0$ with h_0 and its evaluation results
Initialize development-evaluated set $\mathcal{V}_0 \leftarrow \{h_0\}$ and Pareto archive $\mathcal{A}_0 \leftarrow \{h_0\}$
for iteration $n = 0, 1, 2, \ldots$ until budget K is exhausted **do**
    Select candidate-generation context $\mathcal{C}_n^{\text{gen}}$
    $(\mathcal{T}_n^{\text{gen}}, \mathcal{Z}_n^{\text{gen}}) \leftarrow \textsc{CollectEvidence}(h_n, \mathcal{C}_n^{\text{gen}})$
    $h'_n \leftarrow \textsc{GenerateCandidate}(h_n, \mathcal{T}_n^{\text{gen}}, \mathcal{Z}_n^{\text{gen}}, \mathcal{S}_n)$
    Select preliminary screening context $\mathcal{C}_n^{\text{scr}}$
    $(\mathcal{T}_n^-, \mathcal{Z}_n^-) \leftarrow \textsc{Evaluate}(h_n, \mathcal{C}_n^{\text{scr}}, R)$, reusing compatible parent executions when available
    $(\mathcal{T}_n^+, \mathcal{Z}_n^+) \leftarrow \textsc{Evaluate}(h'_n, \mathcal{C}_n^{\text{scr}}, R)$
    $\widehat{p}_n \leftarrow \textsc{BootstrapParetoSupport}(\mathcal{Z}_n^-, \mathcal{Z}_n^+, \epsilon_Q, \epsilon_C, J)$
    if $\widehat{p}_n \geq \gamma$ **then**
        Evaluate h'_n on $\mathcal{D}_{\text{dev}}$ with R repeats and record the results
        $\mathcal{V}_{n+1} \leftarrow \mathcal{V}_n \cup \{h'_n\}$
        $\mathcal{A}_{n+1} \leftarrow \textsc{UpdateParetoArchive}(\mathcal{V}_{n+1})$
    else
        $\mathcal{V}_{n+1} \leftarrow \mathcal{V}_n$
        $\mathcal{A}_{n+1} \leftarrow \mathcal{A}_n$
    end if
    $\mathcal{S}_{n+1} \leftarrow \textsc{UpdateState}(\mathcal{S}_n, h_n, h'_n, \text{all available evidence})$
    $h_{n+1} \leftarrow \textsc{ContinueOptimization}(\mathcal{V}_{n+1}, \mathcal{A}_{n+1}, \mathcal{S}_{n+1}, \pi)$
end for
$\mathcal{H}_\pi^{\text{sel}} \leftarrow \textsc{SelectForDeployment}(\mathcal{V}, \pi)$
return $\mathcal{H}_\pi^{\text{sel}}$

Algorithm 2 ParetoSaddler instantiated on AutoSaddler (adaptations highlighted in orange)

Require: Initial harness h_0, training set $\mathcal{D}_{\text{train}}$, development set $\mathcal{D}_{\text{dev}}$, rollout budget K, number of evaluation repeats R, number of bootstrap draws J, parent-relative local allowances ϵ_Q, ϵ_C, promotion threshold γ, and deployment preference π

Initialize EvoDAG G_0 with h_0
Evaluate h_0 on $\mathcal{D}_{\text{dev}}$ with R repeats and record the results in G_0
Initialize development-evaluated set $\mathcal{V}_0 \leftarrow \{h_0\}$ and Pareto archive $\mathcal{A}_0 \leftarrow \{h_0\}$
for iteration $n = 0, 1, 2, \ldots$ until budget K is exhausted **do**
    Sample mini-batch $\mathcal{B}_n \subset \mathcal{D}_{\text{train}}$
    $(\mathcal{T}_n^-, \mathcal{Z}_n^-) \leftarrow \textsc{Evaluate}(h_n, \mathcal{B}_n, R)$
    Diagnose failures and inefficiencies across repeated executions using $\mathcal{T}_n^-$ and $\mathcal{Z}_n^-$
    Generate a targeted patch $\Delta\theta_n$, yielding candidate $h'_n = h_n + \Delta\theta_n$
    $(\mathcal{T}_n^+, \mathcal{Z}_n^+) \leftarrow \textsc{Evaluate}(h'_n, \mathcal{B}_n, R)$
    $\widehat{p}_n \leftarrow \textsc{BootstrapParetoSupport}(\mathcal{Z}_n^-, \mathcal{Z}_n^+, \epsilon_Q, \epsilon_C, J)$
    if $\widehat{p}_n \geq \gamma$ **then**
        Evaluate h'_n on $\mathcal{D}_{\text{dev}}$ with R repeats and record the results
        $\mathcal{V}_{n+1} \leftarrow \mathcal{V}_n \cup \{h'_n\}$
        $\mathcal{A}_{n+1} \leftarrow \textsc{UpdateParetoArchive}(\mathcal{V}_{n+1})$
    else
        $\mathcal{V}_{n+1} \leftarrow \mathcal{V}_n$
        $\mathcal{A}_{n+1} \leftarrow \mathcal{A}_n$
    end if
    Reflect on $\mathcal{T}_n^-$, $\mathcal{T}_n^+$, and available evaluation results
    Update EvoDAG G_n with the candidate, patch, metrics, and lessons to obtain G_{n+1}
    Evolve h_{n+1} from G_{n+1}, guided by $\mathcal{V}_{n+1}$, $\mathcal{A}_{n+1}$, and π
end for
$\mathcal{H}_\pi^{\text{sel}} \leftarrow \textsc{SelectForDeployment}(\mathcal{V}, \pi)$
return $\mathcal{H}_\pi^{\text{sel}}$

C Additional Analysis for RQ1

Local MinCost Gate. For the gate comparison in Section 5.3, we replace Pareto-aware promotion with a cost-oriented local screening rule within PS-MinCost. This alternative retains repeated execution and Bayesian-bootstrap screening, but requires supported cost reduction rather than progress in either objective.

Using the same parent-relative quality difference $\Delta Q_t^{(j)}$ and relative cost change $\Delta C_t^{(j)}$, the promotion event is

$$E_{t,\mathrm{MC}}^{(j)} = \left\{ \Delta C_t^{(j)} < 0 \ \wedge \ \Delta Q_t^{(j)} \geq -\epsilon_Q \right\}.$$

The joint support and promotion decision are

$$\widehat{p}_{t,\mathrm{MC}} = \frac{1}{J} \sum_{j=1}^{J} \mathbf{1}\left\{ E_{t,\mathrm{MC}}^{(j)} \right\}, \qquad \text{promote } h_t' \iff \widehat{p}_{t,\mathrm{MC}} \geq \gamma.$$

This rule retains only the cost-improvement branch of Pareto promotion: it permits cost reductions with bounded quality loss but does not credit quality gains without cost reduction.

GAIA2 analysis. We complement the ClawMark analysis in Section 5.3 with case studies from the GAIA2 PS-Pareto run. We focus on I-9 and I-17, whose mini-batches achieve perfect quality.

Iteration I-9 achieves perfect quality on all three mini-batch tasks across all three executions before applying patches. Nevertheless, the execution traces expose unnecessary retrieval overhead: successful aggregation workflows retain full email bodies and perform individual contact lookups. I-9 then introduces compact, date-filtered email reads and batched contact lookup, reducing the amount of information retrieved and the number of separate lookup operations required. After the update, screening quality remains at 100%, while mean cost decreases from \$0.824 to \$0.494 per task, by 40.02%. Since every observed quality score is 100% for both harnesses, the quality-only gate would reject the promotion because it requires a strict quality improvement. The Pareto gate instead assigns promotion support of 1.0 and advances I-9 to development evaluation. On the development set, I-9 improves both objectives relative to its parent: quality increases from 54.36% to 55.90%, while cost decreases from \$0.64 to \$0.54 per task, leading to a better development operating point.

I-17 provides another example involving both evidence access and execution control. Its parent again succeeds in all nine screening executions, but successful trajectories still contain inefficient access to email history and unnecessary waiting after task completion. The update adds recency information to summaries, supports oldest-first email retrieval, and introduces termination control to avoid continued idle waiting when no follow-up work remains. Screening quality stays at perfect quality, while mean cost decreases from \$0.50 to \$0.30 per task, a 39.96% reduction. A strict quality-only gate will reject the promotion, whereas Pareto promotion support is 1.0. Development evaluation subsequently shows a quality increase from 56.92% to 60.00%, alongside a cost reduction from \$0.38 to \$0.37 per task. The benefit observed on the screening tasks therefore extends beyond lower local execution cost to a stronger quality–cost operating point on development tasks.

Together with the ClawMark case, these examples show that local quality saturation does not imply that a harness has exhausted its improvement opportunities. Cost supplies an additional promotion signal, allowing updates to retrieval and execution control to receive development evaluation even when screening quality cannot increase.

Development-set comparison under cost minimization. Figure 4 compares PS-MinCost and MH-MinCost on the development sets of ClawMark and GAIA2. On both benchmarks, PS-MinCost identifies lower-cost harnesses with higher development quality than the evaluated MH-MinCost candidates. These results complement the held-out comparisons in Table 1.

D Additional Analysis for RQ2

Variation of Response Quality and Cost across Executions. Execution randomness can affect both objectives. Figure 5 illustrates this effect for a fixed harness evaluated six times on four randomly sampled ClawMark tasks. Variability is also evident during optimization: across the 50

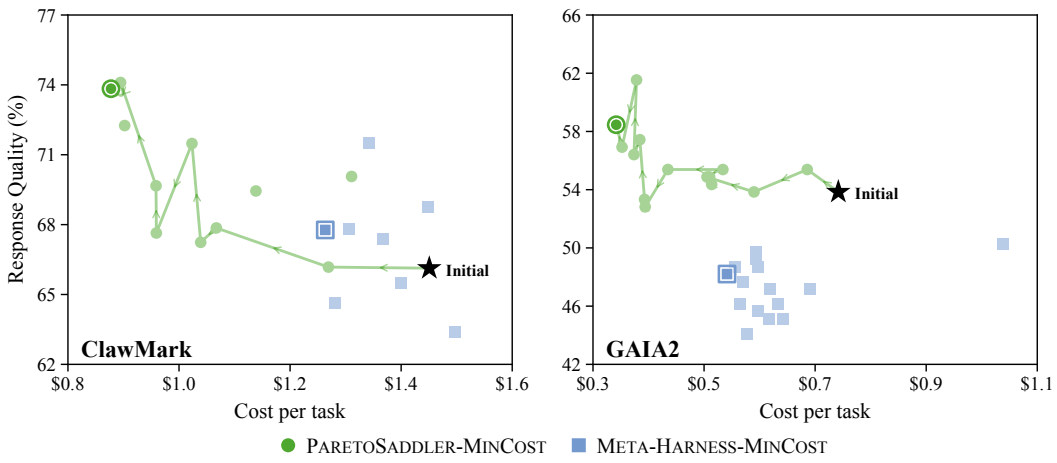

Figure 4: **Quality–cost outcomes under quality-constrained cost minimization on the development sets of ClawMark and GAIA2.** Each marker represents a harness evaluated on the development set; better solutions lie toward the upper left. Stars denote the initial harness, and enlarged outlined markers highlight the final candidates selected by each method. Solid green lines trace harness evolution in PS-MINCOST; green markers not connected by solid lines denote candidates not selected for further refinement. The selected PS-MINCOST harnesses achieve substantially lower cost than their MH-MINCOST counterparts while also attaining higher quality.

training tasks in the PS-PARETO run, the mean per-task coefficient of variation (CV) is 20.35% for quality and 23.30% for cost before patching, and remains 17.05% and 21.86%, respectively, after patching. Across the 20 development tasks, the corresponding CVs are 18.8% for quality and 25.6% for cost. These observations motivate repeated execution when assessing candidate updates. We therefore use R executions per evaluation case during both screening and development evaluation.

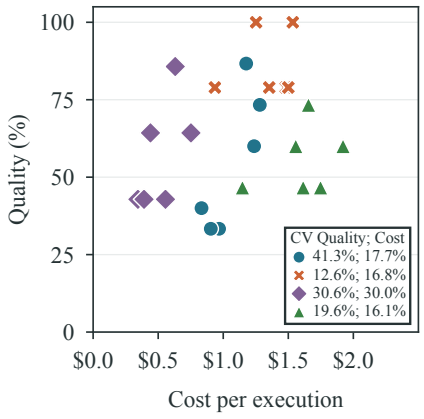

Figure 5: Run-to-run variability on four sampled ClawMark tasks. Markers show six executions per task; the legend gives within-task CV for quality and cost.

Execution-Level Diagnostic Evidence. Table 7 expands the representative ClawMark cases discussed in Section 5.3. Repeated executions provide useful contrasts even when aggregate task outcomes alone appear uninformative. When all runs succeed, differences in execution behavior can expose avoidable overhead, as illustrated by intermittent subagent delegation. When outcomes differ across runs, the contrast reveals trajectory-dependent omissions that may be invisible from a single successful execution. Conversely, failures recurring across all observed runs provide stronger evidence of a systematic defect requiring a harness-level update.

Table 7: Representative examples of how repeated executions inform harness optimization on ClawMark. P/F denotes pass/fail of the named check.

Repeated evidence	**Diagnostic insight**	**Patch and repeated verification**
Costly successful path. A research audit scores 1.0 in all three runs, but costs \$0.72 / \$1.58 / \$0.69. The expensive run makes three subagent-spawn calls; the others make none.	Passing scores conceal different execution strategies. The cheaper local runs provide concrete alternatives to the delegated path; a single passing run would not reveal this contrast.	Prefer local execution for bounded reviews, narrow tool discovery, and disable unavailable web tools. After the patch, all three runs still score 1.0, at \$0.71 / \$0.63 / \$0.81. This is consistent with reduced overhead, not an isolated causal estimate of delegation cost.
Medication omission. The medication-discrepancy check yields P/F/F: the first execution passes, but the next two omit the required finding.	One successful trajectory would hide inconsistent preservation of medication-timeline evidence as a distinct discrepancy.	Preserve one structured row per independent evidence issue. Post-patch failures fall from $2/3$ to $0/3$. A different clinical check fails once after the patch, so targeted repair does not imply complete task correctness.
Customer-risk omission. The customer-champion-loss check yields P/P/F: the missing finding becomes visible only in the third execution.	Combining findings into a generic risk note can lose an independent customer-champion issue, even when other executions retain it.	Keep independent findings as separate evidence rows rather than merging them into one summary. Post-patch failures fall from $1/3$ to $0/3$.
Persistent claim-handling failure. The taxi-claim amount check yields F/F/F, revealing a recurring problem across all observed executions.	The diagnosis identifies eligible missing-receipt claims being omitted or rejected instead of retained for policy-permitted approval.	Add a policy-conditional path that retains eligible pending claims with supporting evidence and requests approval. Post-patch failures fall from $3/3$ to $2/3$.

These examples are descriptive execution evidence; because each patch may modify multiple harness behaviors and only three executions are observed, they should not be interpreted as isolated causal estimates of individual edits.

Table 8: Test results for the single-run and three-run PS-PARETO ablation on ClawMark. Quality is reported in percentage (%), and cost in USD per task. Values are mean $\pm$ sample standard deviation across three test repeats. Superscripts Q and C denote the quality- and cost-extreme harnesses selected on the development set, respectively. **Bold** indicates the best mean result in each column.

Setting	E-commerce (8)		Journalist (8)		Legal Assistant (6)		**Overall (22)**	
	Qual. ↑	Cost ↓	Qual. ↑	Cost ↓	Qual. ↑	Cost ↓	**Qual. ↑**	**Cost ↓**
Single-run^C	72.46 ± 9.90	0.89 ± 0.02	82.34 ± 3.13	**1.05** ± 0.09	75.74 ± 5.15	0.88 ± 0.09	76.95 ± 5.50	0.95 ± 0.06
Single-run^Q	65.87 ± 4.11	0.83 ± 0.06	85.70 ± 1.57	1.19 ± 0.10	75.51 ± 7.50	0.87 ± 0.09	75.71 ± 2.10	0.97 ± 0.05
Three-run^C	72.99 ± 4.29	**0.81** ± 0.01	**89.88** ± 3.18	1.12 ± 0.16	74.72 ± 2.95	**0.83** ± 0.09	79.60 ± 2.43	**0.92** ± 0.08
Three-run^Q	**77.77** ± 7.51	0.87 ± 0.07	87.81 ± 3.55	1.28 ± 0.03	**77.51** ± 5.88	**0.83** ± 0.08	**81.35** ± 3.11	1.01 ± 0.02

Single-Run Ablation Protocol. To test whether the additional execution evidence affects the optimization outcome, we compare the default three-run setting ($R = 3$) with a single-run variant ($R = 1$) of PS-PARETO. With $R = 1$, the bootstrap weights are deterministic, so promotion reduces to checking the local quality–cost condition on the single-run observations. The initial harness, task splits, optimizer, deployment setting, and remaining optimization hyperparameters are held fixed; only the number of executions used to construct and evaluate each update is changed. Consequently, the ablation measures the end-to-end effect of optimizing with repeated evidence, including its influence on diagnosis, patch verification, and subsequent promotion decisions. As reported in Table 8, the three-run search reaches stronger held-out quality–cost operating points at both selected Pareto extremes. The comparison does not isolate repeated evidence from its additional execution cost: using $R = 3$ necessarily allocates more task executions than $R = 1$. We therefore interpret the ablation as evaluating the optimization protocol enabled by repeated execution, rather than as a compute-matched estimate of the marginal value of each additional rollout.

E Additional Analysis for RQ3

Mean-based screening. We compare Bayesian screening with a mean-based gate using the same screening evidence and local allowances. Following Section 4, let $h_t^- = h_t$ and $h_t^+ = h_t'$ denote the parent and candidate, respectively. Using the empirical estimates defined in Section 3, we write

$$\widehat{\Delta Q}_t = \widehat{Q}_{D_t^{\mathrm{scr}}}(h_t^+) - \widehat{Q}_{D_t^{\mathrm{scr}}}(h_t^-), \qquad \widehat{\Delta C}_t = \frac{\widehat{C}_{D_t^{\mathrm{scr}}}(h_t^+) - \widehat{C}_{D_t^{\mathrm{scr}}}(h_t^-)}{\widehat{C}_{D_t^{\mathrm{scr}}}(h_t^-)}.$$

Thus, $\widehat{\Delta Q}_t$ measures the absolute quality difference, while $\widehat{\Delta C}_t$ measures the relative cost change. The mean-based gate promotes h_t' when

$$\left[\widehat{\Delta Q}_t > 0 \wedge \widehat{\Delta C}_t \le \epsilon_C\right] \vee \left[\widehat{\Delta C}_t < 0 \wedge \widehat{\Delta Q}_t \ge -\epsilon_Q\right].$$

Both gates use $\epsilon_Q = 0.10$ and $\epsilon_C = +\infty$.

Table 9 reports repeated observations for archived candidate I-12 and its parent. The mean-based gate would promote I-12, whereas its bootstrap promotion support of 0.55 falls below $\gamma = 0.7$. The end-to-end gate ablation in Section 5.3 separately evaluates how the choice of screening rule affects the resulting optimization outcomes.

Table 9: Repeated quality–cost observations for I-12 and its parent. Brackets contain the three training executions for each task. Differences are computed from the corresponding mean observations as candidate minus parent; quality differences are reported in percentage points (pp).

Task	Parent		Candidate		Difference	
	Quality (%)	Cost ($)	Quality (%)	Cost ($)	ΔQ (pp)	ΔC ($)
Content operation	[17, 100, 100]	[2.89, 1.26, 1.11]	[100, 100, 100]	[1.44, 2.08, 1.94]	+27.67	+0.067
Research assistant 4	[100, 100, 100]	[0.65, 1.34, 1.69]	[13, 67, 87]	[0.76, 0.76, 0.91]	−44.33	−0.417
Research assistant 9	[100, 100, 100]	[0.95, 0.53, 0.65]	[100, 100, 100]	[1.03, 0.63, 0.72]	0.00	+0.083
Overall mean	90.74	1.230	85.19	1.142	−5.55	−0.088

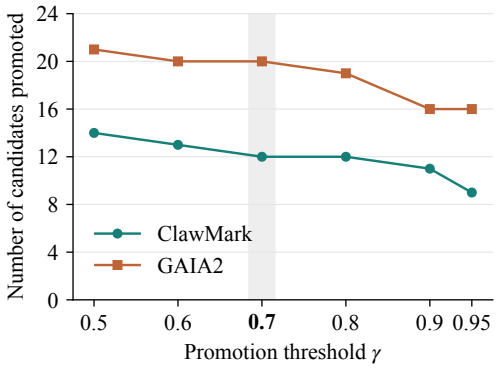

Figure 6: **Retrospective sensitivity to the promotion threshold γ.** Each point shows the number of archived proposals that would receive development evaluation under the corresponding threshold. The shaded region marks the setting used in our experiments.

Sensitivity to the promotion threshold. We replay archived proposals from both benchmarks under different promotion thresholds γ (Figure 6). The setting used throughout our experiments, $\gamma = 0.7$, lies in a locally stable region on both benchmarks, suggesting that it provides a reasonable default. The threshold controls how much statistical support is required before allocating broader evaluation and does not alter the underlying quality–cost preference. For a substantially different deployment domain, however, a small grid search over γ using training or development evidence may be beneficial when additional optimization budget is available.

F Optimization Stability

Because each full optimization run requires substantial computation, we conduct one additional PS-PARETO optimization run on GAIA2 under the same experimental configuration. Table 10 compares the original run reported in the main text (Run 1) with this additional run (Run 2). For each run, we select the quality- and cost-extreme harnesses on the development set and evaluate with three repeated executions. Both runs improve quality and reduce cost relative to the default agent across all three held-out universes. Overall quality ranges from 61.44% to 62.89%, with costs of \$0.44–\$0.47 per task, compared with 56.22% and \$0.78 for the default agent. Both runs also outperform all evaluated baselines in overall mean quality and cost at both development-selected endpoints. These results provide evidence that the quality–cost gains persist across optimization runs.

Table 10: GAIA2 test results across two PS-PARETO optimization runs. Quality is reported in %, and cost in USD per task. Values are mean ± sample standard deviation across three test repeats of each fixed harness. Superscripts Q and C denote development-selected quality- and cost-extreme harnesses. The shared default-agent evaluation is shown once.

Run	Setting	Metric	**Overall**	**GAIA2 by Universe**		
			GAIA2 (300)	U21 (107)	U22 (112)	U27 (81)
–	Default Agent	Qual. ↑	56.22 ± 1.07	58.26 ± 1.08	53.57 ± 1.55	57.20 ± 2.57
		Cost ↓	0.78 ± 0.08	0.82 ± 0.18	0.71 ± 0.03	0.82 ± 0.04
Run 1	PS-PARETO^C	Qual. ↑	62.11 ± 2.46	63.24 ± 4.80	59.23 ± 1.03	64.61 ± 3.77
		Cost ↓	0.47 ± 0.00	0.44 ± 0.01	0.49 ± 0.02	0.47 ± 0.00
	PS-PARETO^Q	Qual. ↑	62.89 ± 2.14	61.99 ± 2.35	60.71 ± 2.36	67.08 ± 1.89
		Cost ↓	0.45 ± 0.02	0.39 ± 0.01	0.49 ± 0.03	0.46 ± 0.02
Run 2	PS-PARETO^C	Qual. ↑	61.44 ± 1.84	60.12 ± 1.08	61.61 ± 6.25	62.96 ± 4.28
		Cost ↓	0.46 ± 0.02	0.38 ± 0.01	0.47 ± 0.03	0.53 ± 0.02
	PS-PARETO^Q	Qual. ↑	62.22 ± 1.84	61.06 ± 3.28	61.61 ± 3.22	64.61 ± 1.43
		Cost ↓	0.44 ± 0.02	0.40 ± 0.03	0.45 ± 0.04	0.49 ± 0.01

G Optimization Efficiency

Optimizer-Side Usage. Table 11 reports optimizer-side resource usage for the GAIA2 experiments. Patch counts are totals for each optimization run, while runtime, estimated monetary cost, LLM calls, and token usage are averaged per generated patch. Fresh-input, cache-read, and output tokens are reported separately. PS-PARETO incurs higher estimated optimizer cost per patch than AS-QUALITY and MH-PARETO (\$9.62 versus \$6.01 and \$5.42, respectively). This additional overhead accompanies stronger held-out results: both development-selected PS-PARETO endpoints achieve higher mean quality at lower execution cost than all evaluated baselines. In particular, PS-PARETO^Q achieves 62.89% quality at \$0.45 per task, compared with 57.33% at \$0.61 for AS-QUALITY and 54.33% at \$0.83 for MH-PARETO^Q. Similarly, PS-MINCOST uses more optimizer cost per patch than MH-MINCOST (\$8.02 versus \$3.58), but achieves higher test quality (56.11% versus 39.78%) at lower execution cost (\$0.44 versus \$0.62 per task).

Table 11: **Optimizer-side resource usage on GAIA2.** Generated patches are counted per optimization run. All remaining quantities are averaged per generated patch.

Method	Generated patches	Time (s) /patch	Est. cost (\$) /patch	LLM calls /patch	Output tokens /patch	Fresh-input tokens /patch	Cache-read tokens /patch
AS-QUALITY	25	604	6.01	63.4	47,331	391,870	5,260,513
MH-MINCOST	15	610	3.58	67.6	36,055	225,106	2,749,406
MH-PARETO	15	619	5.42	103.3	50,100	328,311	4,544,512
PS-MINCOST	25	967	8.02	87.4	63,577	489,543	7,339,602
PS-PARETO	25	982	9.62	86.6	68,526	616,012	8,968,786

H Optimization Trajectories of ParetoSaddler

Figures 7 and 8 visualize the development quality–cost trajectories of PS-Pareto on ClawMark and GAIA2, respectively. Iteration labels identify candidates within each optimization run, while annotations summarize selected harness modifications. All plotted outcomes are development estimates; final held-out results are reported separately in the main text.

ClawMark. The trajectory includes updates to evidence preservation, tool access, and task completion. Early modifications address omitted information and unsuccessful tool interactions. I-7 introduces bounded probing and local-first execution, improving development quality while reducing cost relative to its parent. Later candidates add guidance for audit artifacts, budget exceptions, and independent evidence rows. The final development Pareto archive contains I-11, I-15, and I-16, retaining multiple quality–cost alternatives for deployment selection.

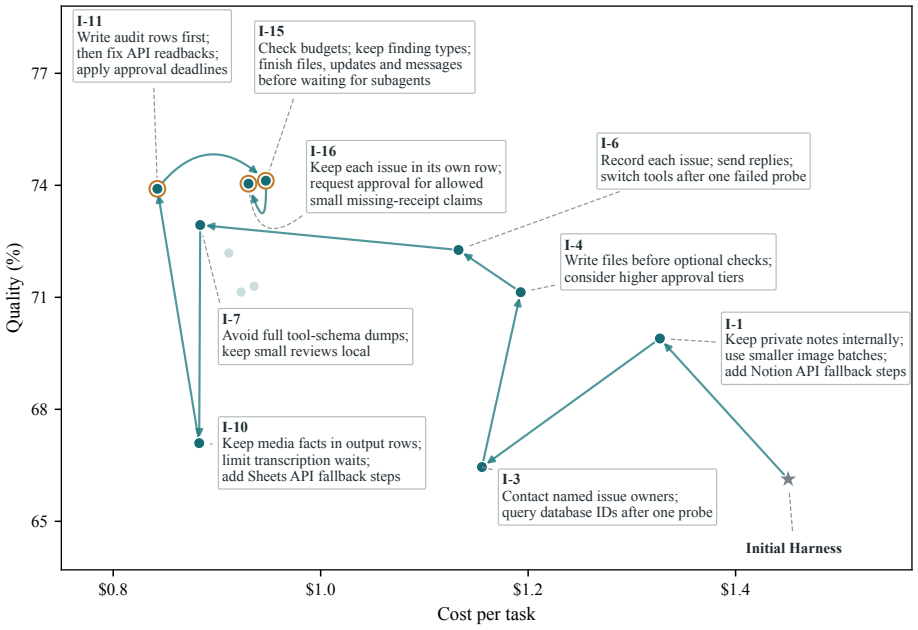

Figure 7: **Annotated development-set optimization trajectory of ParetoSaddler on ClawMark.** Solid arrows trace harness evolution; outlined markers identify the final development Pareto set. Labels indicate proposal iterations, and annotations summarize selected harness modifications.

GAIA2. The annotated updates address information retrieval, ambiguity resolution, and execution control. I-9 reduces retrieval overhead through compact email reads and batched contact lookup, while I-17 improves access to email history and limits unnecessary waiting after task completion. As detailed in Appendix C, both candidates retain perfect screening quality while reducing cost and subsequently improve both development objectives. These cases illustrate how cost reductions can provide a useful promotion signal when local quality is saturated.

The trajectories show that successive updates need not improve both objectives simultaneously, and a candidate can become dominated as later alternatives emerge. The annotations describe implemented changes rather than isolated causal explanations of performance: patches may modify several behaviors, and execution outcomes are stochastic. The controlled gate ablation in Section 5.3 provides complementary evidence about the effect of the promotion rule.

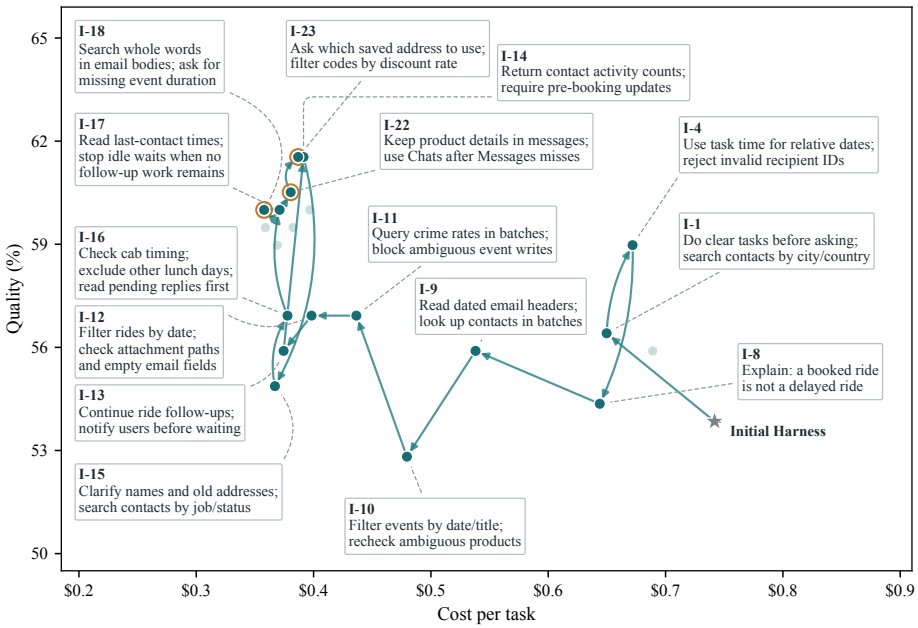

Figure 8: **Annotated development-set optimization trajectory of ParetoSaddler on GAIA2.** Solid arrows trace harness evolution; outlined markers identify the final development Pareto set. Labels indicate proposal iterations, and annotations summarize selected harness modifications.

I Complete ClawMark Optimization Run

We document all 17 proposals from one ClawMark PS-Pareto run of ParetoSaddler. The development split contains 20 tasks, yielding 60 rollouts per full-development evaluation. Table 12 summarizes the optimizer's trajectory-based diagnoses and implemented changes. These diagnoses are interpretations of execution feedback rather than independently established causal explanations. Of the 17 proposals, 12 received full-development evaluation and five were rejected. The final archive contains I-11, I-15, and I-16, with I-16 selected as the final current candidate.

Table 12: Complete ClawMark multi-objective search trace. Rejected candidates did not receive full-development evaluation during optimization. Dominated and Pareto refer to final development status within this search, not status when first evaluated. Prompt and skill guidance describes intended behavior, not an enforced execution guarantee.

Iter	Diagnosed issue	Implemented change	Final status
1	Private allergy notes and same-name ambiguity were overlooked; schema probing, unavailable Python commands, and large image batches consumed budget.	Added internal-evidence and Python compatibility guidance, Notion raw-API fallback steps, and smaller-batch advice in the image-tool description.	Dominated
2	Live trackers were not updated; budgets omitted actual spending; release conclusions omitted observed log gaps; schema and package probing continued after sufficient evidence.	Added guidance for live updates, complete budget arithmetic, release-evidence preservation, and fallbacks without extra dependencies.	Rejected

Continued on next page

Table 12 continued

Iter	Diagnosed issue	Implemented change	Final status
3	Required mentor contact was omitted; repeated Notion schema probes continued despite known identifiers and usable raw APIs.	Added responsible-party contact guidance and instructions to use known identifiers or raw APIs after one schema probe.	Dominated
4	Optional presentation and media exploration delayed required artifacts; a higher compensation tier was conflated with a compliance hold.	Added artifact-first guidance and instructions to report provisional tier eligibility separately while preserving compliance holds.	Dominated
5	Missing memory tools and helpers were repeatedly probed; a disputed appeal was prematurely marked resolved.	Added bounded-fallback and unresolved-appeal guidance.	Rejected
6	Contradictions and independent expense flags were omitted; requested financial messages were only drafted; missing helpers triggered retries.	Added row-level discrepancy and communication guidance, and bounded helper retries.	Dominated
7	Unavailable web tools, full-schema reads, helper probes, and delegation of bounded local work added overhead despite high training scores.	Disabled unavailable web tools in configuration; revised Notion and tool-access skills to avoid full-schema reads; added local-first and bounded-fallback guidance.	Dominated
8	Optional web exploration continued despite sufficient local evidence; one rollout exhausted its length budget without delivery; scheduling, risk calibration, and transcription polling were inefficient.	Added local-evidence, date, and risk-calibration guidance; revised Whisper skill instructions to start with a small model and limit polling.	Dominated
9	Time-zone and commute constraints were not jointly checked; optional meeting setup delayed output; invitations violated an explicit eligibility threshold.	Added scheduling and hard-threshold guidance, artifact-before-setup instructions, and local handling of bounded work.	Dominated
10	Visual facts and unassigned actions were lost; transcription and process-control loops displaced deliverables; live spreadsheet writes were difficult to verify.	Added media-evidence and live-readback guidance, a Sheets REST fallback in tool-use instructions, and targeted transcription with bounded polling.	Dominated
11	Explicit attendance evidence remained pending; backend exploration delayed required audit and reconciliation tables.	Added local-audit-first guidance, followed by live writes and readback, and policy-timestamp-based status decisions.	Pareto
12	Bounded record review was delegated, and execution yielded before required local artifacts were written.	Reworked subagent-orchestration guidance to prioritize local artifacts, delegate only separable work, and yield only when necessary.	Rejected

Continued on next page

Table 12 continued

Iter	Diagnosed issue	Implemented change	Final status
13	HR outlier counts and current recruiting stages were inconsistent; an explicitly required action remained only in notes.	Added outlier-counting, live/local consistency, and independent action-row guidance for explicit instructions.	Rejected
14	Optional backend and public queries continued after sufficient evidence; date discrepancies and budget overruns were not reported as exceptions.	Added evidence-sufficiency stopping guidance and explicit cross-document date, status, and budget discrepancy reporting.	Rejected
15	Budget flags were omitted; waiting for subagents blocked email delivery; generic summaries replaced schema-specific findings.	Added budget-exception and schema-specific evidence guidance; required feasible artifacts, live writes, and communications before yielding.	Pareto
16	An aspirin timeline and a stakeholder's departure were not preserved as independent evidence; a low-value taxi claim without a receipt was rejected despite an approval exception.	Added schema-aligned evidence-row guidance and instructions to request approval for eligible missing-receipt claims rather than silently reject them.	Pareto
17	Recommendations were confused with purchase authorization; primary blockers were omitted; unavailable helpers and invalid Notion URLs disrupted execution.	Added recommendation-versus-authorization and blocker-preservation guidance, with immediate fallback after invalid Notion URLs.	Dominated

Final Harness Changes. We summarize the cumulative changes from the initial harness to I-16, the latest candidate in the final Pareto archive. Table 13 covers six skill and source file changes, grouped by directory. All paths are relative to the OpenClaw repository root.

Table 13: Cumulative harness-file changes in ClawMark I-16 relative to the initial harness. File paths are relative to the directory shown in each group heading.

File	Summary of changes
Tool-use skills `skills/`	
`gog/SKILL.md`	Adds direct Google Sheets REST API fallback when helper tools or packages are unavailable; requires reading back updated ranges to verify live writes.
`mcporter/SKILL.md`	Discourages full schema dumps; recommends compact tool listings or inspection of one specific function.
`notion/SKILL.md`	Adds known mcporter calls and raw-API fallback for task-provided database IDs; requires querying records before concluding and retaining relevant private evidence in internal outputs without unauthorized external disclosure.
`openai-whisper/SKILL.md`	Recommends a small model for short English audio, targeted audio windows for long recordings, one active transcription strategy, and bounded polling before abandoning stalled jobs.

Continued on next page

Table 13 (continued)

File	Summary of changes
Agent and tool prompts `src/agents/`	
`system-prompt.ts`	Prioritizes required artifacts, live updates, read-back verification, and communications before optional exploration or delegation waits. Bounds helper/schema probing and media processing; favors local completion of bounded reviews. Preserves schema-specific evidence and separate independent issues, including medication timelines and customer/champion risks. Clarifies budget checks, policy-based statuses, approval tiers, and pending-approval handling for eligible missing-receipt claims.
`tools/image-tool.ts`	Revises image-tool descriptions to recommend small, focused batches and smaller retries after weak results or timeouts; leaves image-processing implementation unchanged.

Complementary Case: PS-MINCOST. We additionally summarize the selected harness from the ClawMark PS-MINCOST run using the local Pareto-aware gate. Its selected candidate, also indexed I-16, belongs to a separate optimization run from the PS-PARETO candidate above. Table 14 summarizes its cumulative changes relative to its initial harness across 15 skill and source files.

Table 14: Cumulative harness-file changes in ClawMark PS-MINCOST I-16 under Pareto local promotion. File paths are relative to the directory shown in each group heading.

File	Summary of changes
Tool-use skills `skills/`	
`mcporter/SKILL.md`	Discourages repeated full-schema inspection; recommends focused calls and alternative Notion endpoints or raw REST after invalid-URL failures.
`notion/SKILL.md`	Adds sandbox-specific REST fallback guidance and directs batched operations toward supplied resource IDs rather than repeated searches or ID-format retries.
`openai-whisper/SKILL.md`	Prioritizes existing transcripts and small transcription models; limits transcription to decision-critical audio and recommends one long wait instead of repeated polling or duplicate jobs.
`video-frames/SKILL.md`	Recommends an initial contact sheet of four to six frames per video, followed by targeted inspection when needed; preserves visual risks, blockers, ownerless tasks, and delays as structured findings.
Agent tools and execution `src/agents/`	
`agent-tools.read.ts`	Filters detected raw binary text and removes inline image blocks exceeding 24 KiB of base64 data from read results; directs the agent to appropriate extraction or image tools. This changes evidence delivery rather than image resolution.
`sessions/tools/read.ts`	Revises read-tool guidance to favor text files and route documents, images, and other media through dedicated extraction paths.
`bash-tools.exec-output.ts`	Compacts recognized MCP schema dumps of at least 20,000 characters into at most 80 function signatures; appends targeted recovery hints for Notion URL errors and missing Python packages.
`bash-tools.exec.ts`	Detects local Whisper commands and enforces at least a 45-second initial wait before background continuation; adds guidance for decision-critical transcription waits.

Continued on next page

Table 14 (continued)

File	Summary of changes
`bash-tools.process.ts`	Applies shared output compaction to process polling and logs; allows Whisper polls up to 120 seconds while retaining the 30-second cap for other commands; discourages repeated empty polls and duplicate transcription.
`bash-tools.whisper.ts`	Adds a shared detector for Whisper CLI commands, module invocations, and inline Python scripts.
`bash-tools.schemas.ts`	Documents the distinct polling limits for ordinary processes and local Whisper jobs.
`bash-tools.descriptions.ts`	Encourages bounded command output, fewer empty polls, and targeted long waits for necessary transcription.
`bash-tools.shared.ts`	Defines a conditional Python compatibility shim and injects it into Docker shell commands, mapping bare `python` to `python3` only when needed.
`bash-tools.exec-runtime.ts`	Integrates the same compatibility shim into POSIX host execution while preserving configured PATH handling.
`system-prompt.ts`	Keeps bounded evidence-to-writeback work local; requires explicit files, backend updates, and communications. Encourages batched reads, transcript reuse, broad initial visual inspection, in-place tracker updates, correct workspace paths, and bounded dependency fallback.

J Complete GAIA2 Optimization Run

We document all 25 proposals from one GAIA2 PS-Pareto optimization run of ParetoSaddler. The development split contains 65 tasks, yielding 195 rollouts per full-development evaluation. Table 15 summarizes the optimizer's trajectory-based diagnoses and implemented changes.

Table 15: Complete GAIA2 search trace. Rejected candidates did not receive full-development evaluation during optimization. Dominated and Pareto refer to final development status within this search, not status when first evaluated.

Iter	Diagnosed issue	Implemented change	Final status
1	Blocked rescheduling delayed safe deletions; announcements omitted details; contact lookup required paging.	Added partial-completion and message guidance, pagination reminders, and city/country contact search.	Dominated
2	Cancellation messages were vague; punctuation-sensitive product search caused retries.	Added self-contained message guidance and normalized fallback search after empty product results.	Dominated
3	Email replies omitted detail and signatures; bare final text caused invalid-answer retries.	Added reply-completeness guidance and a final-answer tool-use guard.	Rejected
4	Relative dates were misinterpreted; contact IDs were used as phone recipients; separate emails were batched.	Added temporal guidance, Calendar/Email contracts, and phone-recipient validation.	Dominated
5	Literal prefix matching missed required files; missing role search prompted broad contact scans.	Added role-aware contact search and multi-file cleanup guidance.	Rejected

Continued on next page

Table 15 continued

Iter	Diagnosed issue	Implemented change	Final status
6	Name ambiguity failed to block dependent writes; final answers incurred retries.	Added ambiguity-aware write guards, focused blocker messages, and tool-based final delivery.	Rejected
7	Unresolved event references prompted speculative alternatives and broad historical searches.	Added exact-event clarification guidance and narrow date/time search preferences.	Rejected
8	A BOOKED cab's nonzero delay estimate triggered unnecessary contingency actions.	Clarified BOOKED versus DELAYED cab semantics.	Dominated
9	Successful aggregation retained full email bodies and used serial contact lookups.	Added compact date-filtered email reads, batched contact lookup, and lightweight membership checks.	Dominated
10	Broad calendar evidence distorted attendee ranking; shopping output obscured variants.	Added scoped calendar search, variant-aware product output, and finalization guards.	Dominated
11	Daily replacements were created per cancellation; location checks were serial; internal markers polluted context.	Added cardinality guards, batched crime lookup, apartment filters, and internal-marker removal.	Dominated
12	Cab locations and meeting references remained ambiguous; email roles and tool arguments were misinterpreted.	Added ambiguity guards, date-filtered ride history, role search, and revised email/checkout contracts.	Dominated
13	Execution stopped before cab contingencies or waited before updates; questions and product searches were mis-scoped.	Added continuation, update-before-wait, and concise-question guidance, plus age-aware product search.	Dominated
14	Timeouts were treated as availability; activity counting required extensive paging.	Added availability-dependent booking guards and compact contact activity summaries.	Dominated
15	Ambiguous names and stale addresses led to unsafe actions; filtered contacts required paging.	Added recipient/address guards and status/profession contact search.	Dominated
16	Cab timing and pending replies were misinterpreted; email sign-offs were unstable.	Added timing, reply-completeness, lunch-conflict, and email-sign-off guidance.	Dominated
17	Completed shopping re-entered idle waits; email queries lacked recency and oldest-first access.	Added termination guidance, recency summaries, and oldest-first email listing.	Dominated
18	Substring matches caused extra email replies; calendar clarification omitted missing duration.	Added scoped whole-word email search and complete conflict-clarification guidance.	Pareto
19	Purchase messages omitted products; gift emails were unstable; known contacts triggered repeated lookups.	Added product-specific announcement, gift-notice, and compact messaging-availability guidance.	Dominated

Continued on next page

Table 15 continued

Iter	Diagnosed issue	Implemented change	Final status
20	Cancellation workflows omitted follow-up; attendance emails lacked detail; broad calendar reads increased cost.	Added follow-up and email-completeness guards and discouraged broad calendar reads.	Rejected
21	Scheduled-call replies completed writes but failed semantic checks due to underspecified wording.	Added a targeted scheduled-call reply guard and aligned tool guidance.	Dominated
22	Fallback emails omitted unanswered-message context; product descriptors were lost; fuzzy searches were redundant.	Added fallback framing, descriptor-preservation guidance, and direct Messages-to-Chats fallback.	Pareto
23	Phone recipients were treated as unavailable; apartment ambiguity was unresolved; discount inspection was serial.	Added apartment-booking guards, direct phone-number messaging, and filtered discount lookup.	Pareto
24	Combined name/city search obscured name ambiguity before destructive or dependent writes.	Added multi-field contact search and a destructive-action ambiguity guard.	Dominated
25	Location ambiguity blocked unrelated work; cab arguments mismatched expected behavior; discount discovery was repeated.	Added cab ambiguity and immediate-ride guidance, delayed-cab continuation, and threshold discount filtering.	Dominated

Final Harness Changes. We summarize the cumulative changes from the initial harness to I-23, the latest candidate in the final Pareto archive. Table 16 covers the 13 modified harness source files, grouped by directory. All paths are relative to `are/simulation/`.

Table 16: Cumulative source-file changes in I-23 relative to the initial harness. File paths are relative to the directory shown in each group heading.

Source file	Summary of changes
Agent control and prompts `agents/default_agent/`	
`base_agent.py`	Adds pre-tool-use guards for contact and destination ambiguity, calendar updates and invitations, email arguments, batched queries, product descriptors, conditional cab booking, and notification ordering.
`prompts/system_prompt.py`	Directs final and clarification messages through the user-interface tool; requires completing clear earlier subtasks before blocked later actions; anchors relative dates to the task or environment time.
`termination_methods/are_simulation.py`	Continues execution after cab-booking acknowledgements when delay or cancellation handling remains pending; adds termination handling for completed shopping workflows requesting no further notifications.
Application tools `apps/`	
`email_client.py`	Adds date filters, header-only results, oldest-first listing, scoped whole-word search, and batched contact email activity and recency summaries; refines email-sending instructions.
`contacts.py`	Supports multiple contact queries in one search and extends matching to city, country, status, and more precise job or role information.

Continued on next page

Table 16 (continued)

Source file	Summary of changes
`calendar.py`	Adds time-bounded tag lookup and event search with optional title-only matching; refines calendar-operation instructions.
`messaging_v2.py`	Adds batched contact activity summaries, strengthens phone-number validation to reject internal contact IDs, and clarifies recipient lookup, pagination, and message-content requirements.
`messaging.py`	Clarifies pagination limits and message composition, including concise requested questions and preservation of event details and product-variant descriptors.
`shopping.py`	Extends product search to variant attributes with token normalization and age matching; revises product and item representations; adds discount-code filtering by percentage.
`apartment_listing.py`	Adds unsaved-property filtering and multi-zip-code search; instructs the agent to treat multiple matching saved properties as ambiguous for location-dependent actions.
`cab.py`	Adds pagination and date filtering to ride-history lookup; revises ride representation and guidance for quotations, bookings, and status checks, including destination ambiguity.
`city.py`	Supports batched crime-rate queries for multiple zip codes.
`system.py`	Clarifies that required status updates precede waiting and that completed tasks should not end with an unnecessary idle wait.

Complementary Case: PS-MINCOST. We additionally summarize the selected harness from the GAIA2 PS-MINCOST run with the local Pareto-aware gate. Its selected candidate, I-22, belongs to a separate optimization run from the PS-PARETO candidate above. Table 17 summarizes its cumulative changes relative to its own initial harness across 12 Python source files and one runtime hook configuration.

Table 17: Cumulative harness-file changes in GAIA2 PS-MINCOST I-22 with local Pareto promotion. File paths are relative to the directory shown in each group heading.

File	Summary of changes
Agent execution and prompts	
`are/simulation/agents/default_agent/`	
`base_agent.py`	Adds heuristic conversion of terminal plain-text responses into user-interface tool calls after prior tool activity; checks intended to exclude refusals and progress messages limit when conversion applies.
`prompts/additional_prompts.py`	Prioritizes compact search, filtering, and counting over broad listing and repeated lookups; requires tool-based final responses and clarifications; clarifies authorization for simulated app actions.
Application tools `are/simulation/apps/`	
`apartment_listing.py`	Adds multi-zip-code search, unsaved-property filtering, and sorting by square footage; refines listing and saving guidance.
`cab.py`	Adds compact ride-history search by destination, multiple candidate destinations, origin, and status; refines time formatting and booking guidance.
`calendar.py`	Adds compact multi-tag event lookup and attendee-based event counting with date filters; refines event-query and ambiguity-handling guidance.
`city.py`	Adds batched crime-rate lookup and threshold-based zip-code filtering.

Continued on next page

Table 17 (continued)

File	Summary of changes
`contacts.py`	Introduces compact contact representations, batched full-name lookup, age filtering, and exact residence and nationality filters; adds ambiguous-recipient guidance.
`email_client.py`	Adds date, sender, and recipient filters, summary-only results, field-restricted whole-word search, and normalized address matching; revises email-sending and reply instructions.
`messaging_v2.py`	Adds date-window conversation reads and participant-name-aware search; refines recipient lookup, direct reply checking, and product-availability message requirements.
`sandbox_file_system.py`	Adds filtered file discovery by filename prefixes, substrings, or suffixes without returning full directory trees; clarifies directory creation and existence checks.
`shopping.py`	Extends normalized product matching, adds item-specific discount-code lookup, and encourages reuse of available product details and known cart state.
`system.py`	Instructs the agent to execute recurring actions at their scheduled intervals through the final required interval before proceeding to later checks.
Runtime configuration	Repository root
`hook.json`	Adds five pre-tool-use reminders covering recipient ambiguity, direct conversation reads, calendar monitoring, item-specific discounts, and reuse of product details. The calendar reminder references `get_events_added_by_others`, which is not implemented in the archived harness; it is not an added tool capability.

K Optimizer Prompts

Building on AutoSaddler's multi-session optimizer, ParetoSaddler extends the session instructions to account for both response quality and execution cost. We present selected additions and replacements, relative to the corresponding quality-only templates. The boxes reproduce selected passages with line wrapping adjusted; `[...]` marks omissions, and braces denote runtime placeholders. Promotion and archive decisions are computed by the outer optimization loop, rather than by the optimizer's textual assessment.

Candidate Preparation (Session 0). The evolution session uses development-set quality–cost measurements and archive membership to prepare the next candidate. In the template shown here, the outer loop fixes the current parent; the optimizer may repair uncompensated regressions or transfer localized mechanisms from eligible archive members.

Session 0: Candidate Preparation — Modified Instructions

```
## Goal

Prepare the already-forked current harness source tree for this
    iteration's multi-objective optimization patch. The objective is
    to grow the Pareto frontier by maximizing accuracy and minimizing
    cost. The outer loop has already selected the `CURRENT` base;
    Session 0 must not reselect or replace it.

Preparation must be **data-driven**, based on full-development
    objective vectors, MOO status, outcome analysis, execution
    traces, and focused source comparison - not attachment to
    accumulated patches or a single-objective ranking.

[...]
```

```
## Candidate Performance

{candidate_table}

[...]

## Preparation Strategy

The current child is always the starting point. Choose among these
    actions:

- **Keep the current fork unchanged** when no evidence-backed
    Pareto-relevant preparation is warranted.
- **Repair an inherited Pareto regression** by reverting only lines
    that caused quality loss or cost growth without a compensating
    objective gain.
- **Transfer a Pareto mechanism** when a donor has a measured,
    localized quality, cost, or joint improvement that complements
    the current child.

[...]

The resulting source must remain a descendant of the fixed current
    fork. Donor use adds provenance; it does not change the base
    parent.
```

Diagnosis–Patch (Session 1). Diagnosis is extended from unsuccessful tasks to cost-inefficient executions, including tasks that already pass. The session receives measured cost and per-run evidence alongside quality scores, and is instructed to inspect repeated executions before making causal or variance claims. The objective permits quality improvements, cost improvements, and trade-offs, while explicitly guarding against savings obtained by omitting required work.

Session 1: Diagnosis–Patch — Modified Instructions

```
## Goal

Fix incomplete or cost-inefficient scenarios in this mini-batch by
    diagnosing root causes from execution traces and the agent
    codebase, then applying targeted code patches without regressing
    scenarios that already pass.

[...]

The optimization objective is to grow a useful **quality-cost Pareto
    frontier** across the full task distribution, not just this
    mini-batch. Joint improvements, quality-improving, or
    cost-improving are all valid. Patches should address general
    behavioral patterns, not hardcode scenario-specific answers.

[...]

Do not create cheap failure by removing required work or validation.

[...]

## Initial Measured Cost
{cost_listing}

### Cost breakdown - every task
```

```
Token and artifact breakdown for **each** task below. When repeated
    runs are enabled, each task also lists per-run score, cost, token
    counters, and trace paths.
{cost_breakdown}

[...]

### 2. Diagnose failing or expensive scenarios

For each failing scenario or expensive run, use the `diagnose` skill
    to identify the root cause. The key is to find **why** the agent
    made the wrong decision or became costly, not just **what** went
    wrong. Diagnosis requires reading both the execution trace and
    the agent codebase.

Use the exact per-run artifact paths in the cost breakdown to inspect
    harness output. Start with `run/output.jsonl` for score, status,
    rationale, and exception state; then read the matching lite trace
    for agent behavior and input/cache/output token usage. Escalate
    to the matching full trace for
environment, app-state, event, or judge evidence. For `num_runs > 1`,
    inspect every retained run before making causal or variance
    claims; for `num_runs = 1`, the single row and trace are the
    complete run-level evidence. Use only these benchmark artifacts
    for execution and cost attribution.
```

Reflection (Session 2). Reflection separately records measured objective movement, evidence that the intended mechanism was exercised, and attribution to the patch or execution variability. It asks whether additional work supports quality and whether cost reductions preserve required behavior.

Session 2: Reflection — Modified Instructions

```
## Objective history (accuracy up, cost down)

Per-candidate measured cost/task and accuracy so far. The archive
    contains every nondominated vector; it is not ranked into one
    champion.

{objective_history}

### Full-validation interpretation

{full_validation_context}

When C{candidate_idx} has a measured full-validation row:

1. Compare its cost and accuracy with the seed, immediate parent,
    current, and other Pareto archive members.
2. Check whether the same-minibatch cost direction held at
    full-validation scale.
3. Attribute the aggregate movement only to mechanisms supported by
    the patch and available evidence; do not invent per-scenario
    validation behavior.
4. Record the conclusion through `--generalization-note` on the
    relevant scenario reflections. If C{candidate_idx} has no
    measured row, say that generalization is unmeasured rather than
    inferring it from prior candidates.

[...]

Never rewrite these machine-computed decisions from trace
    interpretation.
```

```
Separately classify whether the intended mechanism was exercised and
    worked:

- `supported`: the changed path ran, replaced the diagnosed waste,
    and the measured movement supports the mechanism.
- `partially_supported`: it worked on only part of the path or one
    objective.
- `unsupported`: the path did not run or did not replace the target
    behavior.
- `contradicted`: the path ran but added work or caused the opposite
    behavior.
- `unmeasured`: the current scenarios did not exercise the mechanism.

Also record causal attribution as `patch_caused`, `stochastic`,
    `mixed`, or `uncertain`. Mechanism verdict and causal attribution
    are analysis outputs; objective movement, train screening, and
    MOO status are machine inputs.

[...]

- **`quality_gain_cost_loss`**: Did extra work causally buy the
    quality gain? If development-evaluated, did it produce a
    nondominated admissible point?
- **`cost_gain_quality_loss`**: Did savings come from removing waste
    or from skipping required work? Did the development admit it?

[...]

There are four independent classifications on every command:

- `--status` is the exact-pass transition from the per-scenario table.
- `--movement-label` is the machine-computed scenario movement.
- `--mechanism-verdict` records whether the modified mechanism was
    exercised.
- `--causal-attribution` records whether behavior was patch-caused,
    stochastic,
  mixed, or uncertain.
```

L Limitations and Future Work

Evaluation cost and uncertainty. Repeated executions improve optimization evidence but increase evaluation cost. Our Bayesian bootstrap captures variation among executions on fixed screening tasks, rather than uncertainty from task sampling. Future work could jointly adapt task selection and repetition counts to allocate evaluation effort where promotion decisions remain uncertain.

Beyond Pareto-aware promotion. Our promotion rule provides one operational definition of a promising intermediate candidate. Other local gates could preserve candidates that improve particular task instances or exhibit novel behaviors, even without favorable aggregate quality–cost changes. An LLM-based gate could also use execution traces and optimization history to judge whether a candidate warrants broader development evaluation. Evaluating these alternatives would clarify how to balance immediate gains, potential for later improvement, and screening cost.