JOINT MODEL–HARNESS OPTIMIZATION: COST-AWARE RESOURCE ALLOCATION FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

An LLM agent's quality and cost depend jointly on its model and harness, yet execution-time gains alone do not establish whether harness optimization is cost-effective. We study model–harness design as resource allocation, accounting separately for the cost of searching for a harness and the cost of executing it. We introduce HARP-Static, which combines component addition, trace-guided graph rewrites, and backward compression while tuning node-level resources and model assignments. Across six benchmarks, with harness search on five, we find both opportunities and limits to reallocating computation. Two searched architectures reduce to a minimal reasoning chain, and development-set analyses identify verification changes that lower cost without changing observed scores; larger savings from cheaper node-level models incur quality loss. On HotpotQA, Haiku-4.5 with a searched harness outperforms Opus-4.5 with its default harness by 10.4 percentage points at 55.3% of its execution cost. However, the only finite break-even point estimate in the matched-quality analysis requires approximately 1.72 million served tasks, and the underlying saving is not statistically established. Generalization poses a further constraint: on BFCL, the compressed harness loses 15.0 percentage points to the official harness, with post-hoc analysis locating the entire deficit in multi-turn tasks requiring repeated tool interaction. These results show that evaluating harness optimization requires distinguishing local execution gains from deployment-level savings and testing whether compression preserves the interactions a task requires.

1 INTRODUCTION

Large language model (LLM) agents are increasingly built as systems rather than standalone models. Their performance depends not only on the capability of the underlying LLM, but also on the *harness* that organizes reasoning, retrieval, tool use, verification, and recovery. Both stronger models and richer harnesses can improve task performance, but both consume computational resources. This raises a natural question: *how much of a model upgrade can a better harness buy?*

Existing work typically studies these resources separately, through model selection, test-time compute, prompting and scaffolding, tool use, or verification. In deployment, however, these choices compete for the same budget. The central problem is therefore not only how to optimize a harness, but how to allocate computation jointly between model capability and harness capability: given a fixed budget, when should resources be spent on a stronger model, and when should they instead be spent on a better harness? If computation is allocated to the harness, a second question follows: how should that computation be organized across components, execution structure, and node-level resources? We formulate model–harness design as a cost-aware resource-allocation problem.

Our study considers three coupled decisions: model selection, offline harness design, and online resource control. Offline, the harness can adapt its composition, topology, and configuration to task demands; online, it can allocate additional reasoning, retrieval, verification, recovery, or model capacity as the execution state changes. We further ask whether these designs remain effective across tasks and under degraded or failure conditions. This study develops a framework for joint model selection, harness optimization, and adaptive resource allocation.

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

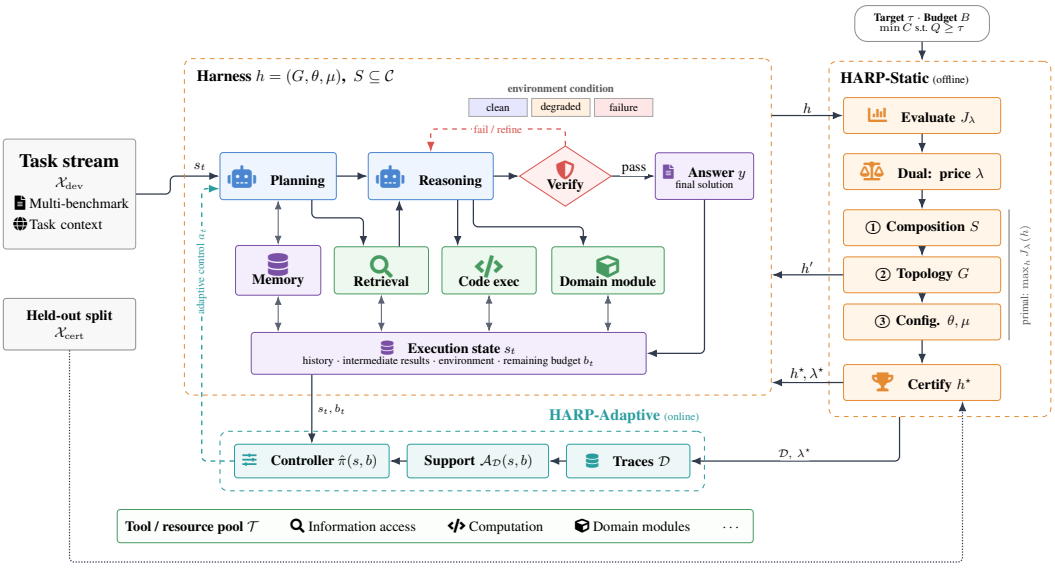


Figure 1: Overview of HARP and the executable harness. The harness $h = (G, \theta, \mu)$ defines the computation graph, node configuration, and model assignment. HARP searches over this design space to obtain an optimized harness under quality and cost constraints.

2 Related Work

LLM agents have evolved from manually engineered reasoning–action loops into increasingly rich execution systems. Early systems such as ReAct and SWE-agent showed that tool interfaces, execution feedback, and agent–environment interaction can substantially change model behavior (Yao et al., 2023; Yang et al., 2024). Modern harnesses extend this role to context management, persistent state, subagents, code execution, verification, and long-horizon control, as exemplified by Deep Agents and the broader harness engineering literature (LangChain, 2026; Tang et al., 2026). In parallel, prompt and program optimization methods such as DSPy and GEPA automate instructions, demonstrations, or textual program parameters while largely preserving the surrounding execution structure (Khattab et al., 2023; Agrawal et al., 2026). These developments shift the design problem from optimizing a single prompt toward optimizing the computational system around the model.

A growing literature therefore treats agent structure itself as searchable. ADAS, AgentSquare, AFlow, and Archon search over agent modules, workflows, or inference-time architectures (Hu et al., 2024; Shang et al., 2024; Zhang et al., 2024; Saad-Falcon et al., 2025). Recent work moves further toward end-to-end harness optimization. Meta-Harness searches harness code using previous candidates, execution traces, and evaluation feedback; AutoHarness synthesizes executable harnesses from environment feedback; and Better-Harness uses evaluation-driven hill climbing over production-style traces (Lee et al., 2026a;b; Lou et al., 2026; Trivedy, 2026). Related open implementations apply the same idea to legal and coding agents (Niklaus, 2026b;a). These studies establish that substantial capability can be gained without changing model weights. However, their central object is typically the improved harness itself: search quality, execution tokens, or call budgets are optimized, but the one-time cost of discovering the harness is not generally separated from its repeated deployment cost or evaluated through a break-even deployment analysis.

A complementary line of work studies how computation should be allocated once an inference system is specified. FrugalGPT and RouteLLM route requests across models to improve quality–cost tradeoffs (Chen et al., 2023; Ong et al., 2024); test-time scaling studies when additional inference computation can substitute for model scale (Snell et al., 2024); and cost-of-pass evaluates model performance directly in economic terms (Erol et al., 2025). Adaptive methods further condition reasoning or verification effort on task difficulty or intermediate state (Damani et al., 2024; Singhi et al., 2025). ToolOrchestra extends this idea to joint model and tool orchestration through a learned efficiency-aware policy (Su et al., 2025). These approaches demonstrate that computation should

not be allocated uniformly, but they generally optimize within a predefined scaffold or action space. Conversely, harness-search methods modify that scaffold offline but do not usually treat the search traces themselves as data for subsequent state-dependent resource allocation.

Evaluation has simultaneously expanded toward realistic, interactive, and resource-intensive settings. τ-bench evaluates tool–agent–user interaction, Terminal-Bench targets long-horizon terminal tasks, and CritPt tests frontier scientific reasoning (Yao et al., 2024; Merrill et al., 2026; Zhu et al., 2025). Frameworks such as DeepEval support trace-level evaluation and regression testing (Confident AI, 2026), while recent analyses show that inference compute can materially change benchmark conclusions (McFadyen et al., 2026). Taken together, existing work has advanced harness search, cost-aware inference, adaptive computation, and agent evaluation, but these components remain largely separated. We study model choice and harness design as a joint resource-allocation problem, explicitly distinguish search cost from execution cost, optimize harness composition, topology, and configuration, and reuse offline search traces for online resource control. This allows us to ask not only whether a harness improves an agent, but when it can substitute for model capability, whether the optimization itself is worth its cost, and whether the resulting allocation generalizes beyond the search distribution.

3 Problem Formulation and Method

HARP casts harness design as a constrained optimization problem and solves it with a primal–dual scheme. A single scalar, the *shadow price of computation* λ, couples the three places where cost is traded for quality: which components a harness retains, how its nodes are configured, and which actions are taken at runtime. Every stage of HARP approximately maximizes the same Lagrangian at a common price. Each stage carries a guarantee whose looseness is governed by a structural constant of the harness space that we estimate from search traces.

3.1 Cost-Constrained Agent Design

We represent an agent by a model–harness pair $z = (m, h)$, where $h = (G, \theta, \mu)$ specifies the execution graph, node-level configuration, and node-level model assignment. Running z on a task $x \sim \mathcal{P}$ yields quality $q(x, z) \in [0, 1]$ and cost $c(x, z) \in [0, \bar{c}]$, with expectations $Q(z)$ and $C(z)$. We seek the least expensive design that reaches a target quality τ:

$$(m^\star, h^\star) \in \arg\min_{m,h} \; C(m, h) \quad \text{s.t.} \quad Q(m, h) \geq \tau. \tag{1}$$

For a fixed model,

$$C_m^\star(\tau) = \min_{h} \; C(m, h) \quad \text{s.t.} \quad Q(m, h) \geq \tau, \tag{2}$$

and $C^\star(\tau) = \min_m C_m^\star(\tau)$. The curve $\tau \mapsto C_m^\star(\tau)$ is the model-conditioned cost–quality frontier.

Lagrangian scalarization. Problem equation 2 is combinatorial, and its constraint is observable only through noisy evaluation. HARP therefore works with the Lagrangian

$$J_\lambda(h) = Q(m, h) - \lambda\, C(m, h), \qquad h_\lambda \in \arg\max_h J_\lambda(h), \qquad \lambda > 0, \tag{3}$$

where λ is the quality a designer is willing to forgo per dollar saved. Proposition 1 shows that sweeping λ is an exact parametrization of the convex part of the frontier.

Proposition 1 (Lagrangian characterization of the frontier)**.** *Fix m and let $\breve{C}_m$ be the lower convex envelope of $C_m^\star$.*

(i) Exactness. *h_λ solves equation 2 at its own quality level, that is,*

$$C(m, h_\lambda) = C_m^\star\big(Q(m, h_\lambda)\big).$$

(ii) Monotonicity. *If $\lambda_1 < \lambda_2$, then $Q(m, h_{\lambda_1}) \geq Q(m, h_{\lambda_2})$ and $C(m, h_{\lambda_1}) \geq C(m, h_{\lambda_2})$.*

(iii) Duality gap. *Every point $\big(Q(m, h_\lambda), C(m, h_\lambda)\big)$ lies on $\breve{C}_m$, and every vertex of $\breve{C}_m$ is attained for some λ. If $\tau \in (Q_1, Q_2]$ for consecutive vertices (Q_1, C_1), (Q_2, C_2) attained*

by h_1, h_2, then h_2 is feasible and $C(m, h_2) - C^\star_m(\tau) \le C_2 - C_1$. Executing h_2 with probability $p = (\tau - Q_1)/(Q_2 - Q_1)$ and h_1 otherwise attains quality τ at expected cost $\breve{C}_m(\tau) \le C^\star_m(\tau)$.

Part (i) certifies that every design HARP returns for a given price is optimal for the quality it achieves. Part (ii) allows the price that meets a target τ to be located by bisection instead of a grid. Part (iii) isolates the only loss introduced by scalarization, the non-convexity of the frontier, and shows that per-task randomization between two adjacent harnesses removes it.

3.2 Structure of the Harness Space

Let h_R be a rich harness with optional components $\mathcal{C}$ (memory, retrieval, code execution, domain modules, verifier). A harness is determined by a retained set $S \subseteq \mathcal{C}$, a topology G over S, and a per-node option $k_v \in \mathcal{K}_v$. Each option fixes both the configuration θ_v and the model μ_v used at node v, and each $\mathcal{K}_v$ contains a default with zero incremental cost and gain. Write $\Delta_i(S) = Q(S \cup \{i\}) - Q(S)$ for the marginal quality of component i under a fixed topology and default configuration. HARP's guarantees rest on three assumptions, each quantified by a constant we measure.

Assumption 1 (Diminishing returns in composition)**.** *Q is monotone in S, cost is modular, $C(S) = c_0 + \sum_{i \in S} c_i$, and Q has submodularity ratio*

$$\gamma = \min_{\substack{L, A \subseteq \mathcal{C} \\ L \cap A = \emptyset}} \frac{\sum_{i \in A} \Delta_i(L)}{Q(L \cup A) - Q(L)} > 0,$$

with $0/0 := 1$. Submodular Q has $\gamma \ge 1$.

Assumption 2 (Near-separable configuration)**.** *Given (S, G), selecting option k at node v adds cost Δc_{vk} and, up to an interaction residual bounded by η, quality Δq_{vk}: $\left| G(y) - \sum_{v,k} \Delta q_{vk} y_{vk} \right| \le \eta$ for every assignment y, where $G(y)$ is the true quality gain.*

Assumption 3 (Bounded i.i.d. outcomes)**.** *Tasks are drawn i.i.d. from $\mathcal{P}$, with $q \in [0, 1]$ and $c \in [0, \bar{c}]$.*

3.3 HARP-Static: A Primal–Dual Harness Optimizer

For a fixed price λ, HARP maximizes J_λ by block coordinate ascent over composition, topology, and configuration. An outer loop then adjusts λ and certifies the result.

Composition. In the forward direction, reaching quality τ at minimum component cost is a submodular-cover problem. HARP adds components by quality gained per dollar:

$$i_t \in \arg\max_{i \notin S_t} \frac{\Delta_i(S_t)}{c_i}, \qquad S_{t+1} = S_t \cup \{i_t\}. \tag{4}$$

Proposition 2 (Cost-effective composition)**.** *Under Assumption 1, let $C^\dagger_\tau = \min\{\sum_{i \in S} c_i : Q(S) \ge \tau\}$, $c_{\max} = \max_i c_i$, and $\tau - Q(S_0) > \epsilon > 0$. Rule equation 4, run from S_0 until $Q(S_T) \ge \tau - \epsilon$, satisfies*

$$\sum_{i \in S_T \setminus S_0} c_i \;\le\; \frac{C^\dagger_\tau}{\gamma} \ln \frac{\tau - Q(S_0)}{\epsilon} \;+\; c_{\max}. \tag{5}$$

If the rule instead stops once $\max_i \Delta_i(S_t)/c_i < \lambda$, no single addition increases J_λ.

For $\gamma = 1$, equation 5 recovers the logarithmic guarantee of greedy submodular cover (Wolsey, 1982). The ratio stopping rule equates the marginal quality per dollar of the last retained component with the price λ, which is the discrete analogue of a complementary-slackness condition.

Remark 1 (One-shot backward pruning). HARP also prunes backward from h_R. It ranks components by their leave-one-out loss $\ell_i = Q(h_R) - Q(h_R^{-i})$ and removes the longest prefix of this ordering whose lower-confidence quality drop is at most ϵ. If the ordering is preserved on every intermediate set, this path coincides with greedy backward elimination with re-ranking, and it needs $O(|\mathcal{C}|)$ rather than $O(|\mathcal{C}|^2)$ evaluations. Because the forward and backward results are compared on J_λ, a violation of rank stability only costs efficiency. We report the rank correlation between one-shot and re-ranked orderings.

Topology. Given S, HARP applies local graph edits $o \in \mathcal{O}(h)$: removing an edge, merging two model nodes, parallelizing independent calls, or relocating the verifier. An edit is promoted only if it improves the Lagrangian with confidence:

$$\mathrm{LCB}\big[J_\lambda(o(h)) - J_\lambda(h)\big] \geq \rho. \tag{6}$$

Lemma 1 (Finite monotone improvement)**.** *On the event that all confidence bounds hold, each promoted edit increases J_λ by at least ρ. Topology search therefore terminates after at most $(1 - J_\lambda(h_1))/\rho$ promotions, at a ρ-local optimum of J_λ with respect to $\mathcal{O}$.*

Configuration and model assignment. Let $y_{vk} \in \{0, 1\}$ indicate that node v uses option k. Under residual budget B_r, configuration is a multiple-choice knapsack problem:

$$\max_y \ \widetilde{G}(y) = \sum_{v,k} \Delta q_{vk} y_{vk} \quad \text{s.t.} \quad \sum_{v,k} \Delta c_{vk} y_{vk} \leq B_r, \qquad \sum_k y_{vk} = 1 \ \ \forall v. \tag{7}$$

Because λ already prices cost, the Lagrangian of equation 7 separates across nodes:

$$k_v(\lambda) \in \arg\max_{k \in \mathcal{K}_v} \ \Delta q_{vk} - \lambda \, \Delta c_{vk}. \tag{8}$$

Node-level model assignment μ_v is therefore chosen jointly with θ_v at no additional search cost.

Proposition 3 (Configuration optimality)**.** *Let $y(\lambda)$ be defined by equation 8.*

(i) $y(\lambda)$ is an exact optimum of equation 7 with $B_r = \sum_{v,k} \Delta c_{vk} y_{vk}(\lambda)$.

(ii) For arbitrary B_r, let $\hat{y} = y(\hat{\lambda})$, where $\hat{\lambda}$ is the smallest price whose solution is feasible. Under Assumption 2, and assuming no ties at $\hat{\lambda}$,

$$G(y^\star) - G(\hat{y}) \ \leq \ \underbrace{\max_{v,k} \Delta q_{vk}}_{\textit{integrality}} \ + \ \underbrace{2\eta}_{\textit{interaction}}, \tag{9}$$

where $y^\star$ maximizes the true gain G under the constraints of equation 7.

The first term is the integrality gap of the single node that is fractional in the LP relaxation. The second term is the only price paid for the additive surrogate. Unlike a bound stated relative to the relaxation value, both terms are computable.

Price selection and certification. By Proposition 1(ii), $Q(h_\lambda)$ is nonincreasing in λ. HARP therefore places candidate prices by bisection on $\log \lambda$, using lower confidence bounds on the development split. Because the inner maximization is approximate, bisection is used only to place candidates; the guarantee below does not rely on monotonicity. The resulting candidates $\mathcal{H}$, pooled over models with $K = |\mathcal{H}|$, are certified once on a disjoint split of n tasks that is never used during search:

$$\begin{gathered} h^\star = \arg\min_{h \in \mathcal{H}} \widehat{C}(h) \quad \text{s.t.} \quad \widehat{Q}(h) - \beta_n \geq \tau, \quad \widehat{C}(h) + \bar{c}\,\beta_n \leq B, \\ \beta_n = \sqrt{\frac{\ln(2K/\delta)}{2n}}. \end{gathered} \tag{10}$$

Proposition 4 (Certified feasibility)**.** *Under Assumption 3, with probability at least $1 - \delta$, the harness returned by equation 10 satisfies $Q(h^\star) \geq \tau$ and $C(h^\star) \leq B$.*

3.4 HARP-Adaptive: Support-Constrained Lagrangian Control

The static harness fixes *which* capabilities exist; the adaptive controller decides *when* to use them. We model execution of $h^\star$ as a decision process of horizon H. The state s_t contains the history, intermediate results, environment condition, and remaining budget b_t. The actions $\mathcal{A}(h^\star)$ include retrieving, executing code, re-planning, verifying, and emitting the answer a_{ans}. The controller optimizes the same Lagrangian as the static stage,

$$\max_\pi \ J_\lambda(\pi) = \mathbb{E}_\pi\Big[q - \lambda \sum_{t=1}^{H} c(a_t)\Big] \quad \text{s.t.} \quad \sum_{t=1}^{H} c(a_t) \leq B, \tag{11}$$

with $\lambda = \lambda^\star$ inherited from HARP-Static, so that offline design and online control price computation consistently.

Support constraint. The traces $\mathcal{D}$ collected during static search record the effect of each action from many intermediate states. HARP-Adaptive takes only actions whose effect has been observed sufficiently often and that are affordable with a safety margin $\kappa \geq 1$:

$$\mathcal{A}_{\mathcal{D}}(s,b) = \big\{a \in \mathcal{A}(h^\star) : \big(n_{\mathcal{D}}(\phi(s),a) \geq n_{\min} \text{ or } a = a^{\mathrm{st}}(s)\big), \\ \kappa\, \bar{c}_a \leq b\big\} \cup \{a_{\mathrm{ans}}\}. \tag{12}$$

Here ϕ is a state abstraction used for counting, $\bar{c}_a$ is the mean cost of a, and $a^{\mathrm{st}}(s)$ is the action the static harness would take. The static action is always supported, because traces of $h^\star$ itself are in $\mathcal{D}$. The controller is greedy with respect to a value estimate fitted on $\mathcal{D}$:

$$\hat{\pi}(s,b) \in \arg\max_{a \in \mathcal{A}_{\mathcal{D}}(s,b)} \widehat{Q}_\lambda(s,a). \tag{13}$$

Restricting to supported actions follows the conservative principle of offline reinforcement learning: $\widehat{Q}_\lambda$ is trusted only where $\mathcal{D}$ provides evidence.

Proposition 5 (Safe adaptivity)**.** *Let $\Pi_{\mathcal{D}}$ be the policies that act in $\mathcal{A}_{\mathcal{D}}$, let $\pi^\star_{\mathcal{D}}$ be the best of them for J_λ, and let $Q^\star_{\lambda,t}$ be its action values. Suppose $|\widehat{Q}_\lambda(s,a) - Q^\star_{\lambda,t}(s,a)| \leq \xi$ for all supported (s,a) and all t.*

(i) $J_\lambda(\pi^\star_{\mathcal{D}}) - J_\lambda(\hat{\pi}) \leq 2H\xi$.

(ii) The budget-filtered static policy π^{st} belongs to $\Pi_{\mathcal{D}}$. If running the controller costs c_{obs} per step, then

$$J_\lambda(\hat{\pi}) \;\geq\; J_\lambda(\pi^{\mathrm{st}}) + \Delta_{\mathrm{adapt}} - 2H\xi - \lambda H c_{\mathrm{obs}},$$

where

$$\Delta_{\mathrm{adapt}} = J_\lambda(\pi^\star_{\mathcal{D}}) - J_\lambda(\pi^{\mathrm{st}}) \geq 0.$$

(iii) If $\Pr[c(a) > \kappa \bar{c}_a] \leq p$ for every supported action and $c(a_{\mathrm{ans}}) = 0$, the budget B is exceeded with probability at most Hp.

Part (ii) turns the question of whether adaptivity helps into a testable inequality: the adaptive controller improves on the optimized static harness exactly when the attainable gain Δ_{adapt} exceeds the value-estimation penalty $2H\xi$ plus the priced observation overhead $\lambda H c_{\mathrm{obs}}$. Δ_{adapt} grows with task heterogeneity and environment degradation, while ξ shrinks with trace coverage. This predicts when adaptivity pays, which we test across the clean, degraded, and failure conditions. Algorithm 1 (appendix B) summarizes both stages.

Empirical diagnostics. Each guarantee is stated in terms of a measurable constant, and we report all of them: $\hat{\gamma}$, estimated on sampled pairs (L, A); the rank correlation behind Remark 1; $\hat{\eta}$, from held-out multi-node configuration changes; $\hat{\xi}$, the held-out error of $\widehat{Q}_\lambda$; and $\widehat{\Delta}_{\mathrm{adapt}}$. This makes it possible to check whether HARP's behavior in a given regime is explained by the regime's structure.

4 Experimental Setup

Experimental protocol. Our experiments separate four effects that are otherwise confounded in agent evaluation: model capability, harness design, runtime adaptation, and distribution shift. We first cross model capability with default versus searched harnesses to measure the joint cost–performance frontier. We then analyze how offline search changes harness composition, topology, and configuration, test whether search traces support adaptive runtime allocation at matched expected cost, and finally evaluate transfer under held-out tasks and degraded or failure conditions.

Tasks and models. We evaluate on SimpleQA, HotpotQA, WikiTableQuestions, GPQA-Diamond, BFCL, and τ-bench, covering factual QA, retrieval, structured computation, difficult reasoning, tool use, and long-horizon interaction. Harness optimization is performed on development splits and frozen before held-out evaluation. Model arms span cheaper, stronger, and cross-vendor choices available through the same execution gateway. Benchmark-specific splits, metrics, resource caps, and perturbations are given in appendix C.

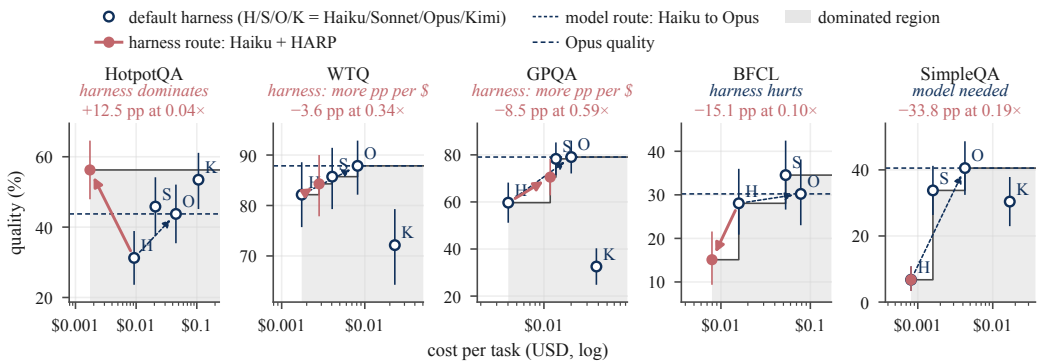


Figure 2: **The harness route is cheaper than the model route, and on procedural tasks it is also better.** Solid clay arrow: Haiku 4.5 with its default harness → with the HARP harness. Dashed navy arrow: the same start → Opus 4.5 with the default harness. Panel header: HARP-Haiku 4.5 minus Opus 4.5, and relative cost. Grey: dominated region. Bars: 95% CIs, paired `test` tasks.

Search and adaptive evaluation. HARP-Static evaluates candidate harness modifications on paired task ids and random seeds, combining forward expansion, graph and configuration refinement, and backward compression. Every search evaluation is logged and counted toward the optimization cost. HARP-Adaptive is trained from these offline search traces and compared against the frozen static harness under matched expected execution cost, using only information available at decision time.

Measurement and statistics. Quality is measured using each benchmark's prescribed evaluation protocol. Execution cost is computed from logged model usage using a single pinned pricing table, including cache-replayed calls at their nominal deployment price. Comparisons are paired by task and seed, with task-clustered bootstrap confidence intervals and multiple-comparison correction where applicable. Cost-matched and quality-matched comparisons are reported only within the measured support; no frontier extrapolation is used.

5 Results

5.1 Model Capability versus Harness Capability

Starting from the cheapest model with its default harness (Haiku 4.5), a practitioner can spend in two ways: upgrade to Opus 4.5 (the model route), or keep Haiku 4.5 and optimize its harness with HARP (the harness route). Figure 2 draws both routes on the measured cost–quality plane, the empirical counterpart of $C^{\star}_{m}(\tau)$ in equation 2. The harness route is cheaper on every benchmark, and on procedural tasks it also reaches higher quality.

HotpotQA shows the clearest case. HARP-Haiku 4.5 exceeds Opus 4.5 by $+12.5$ pp [$+6.3$, $+19.4$] at $0.04\times$ its cost, and it is also cheaper than Haiku 4.5's own default harness. The best default-harness model (Kimi K3, 53.5%) stays below it at $61\times$ the cost. Multi-hop QA is limited by procedure: the searched harness decomposes the question and orchestrates retrieval, which a single call to a stronger model cannot do.

On WTQ and GPQA the harness route raises Haiku 4.5 from 82.1 to 84.3% and from 59.7 to 70.5%, at $0.34\times$ and $0.59\times$ the cost of Opus 4.5. Each extra dollar buys more quality on this route than on the model route. Opus 4.5 still reaches higher absolute quality (87.9% and 79.1%), and neither gap is significant after Holm correction, so the harness is the efficient first step and a model upgrade adds the last few points.

The harness route fails in two cases. On SimpleQA it leaves quality unchanged (6.8%), because closed-book facts come from the model. On BFCL the HARP harness is cheaper but worse (28.1% → 15.1%), which we trace to its tool-call budget in section 5.2. Procedure favors the harness and knowledge favors the model, so the bottleneck should be diagnosed before choosing where to spend.

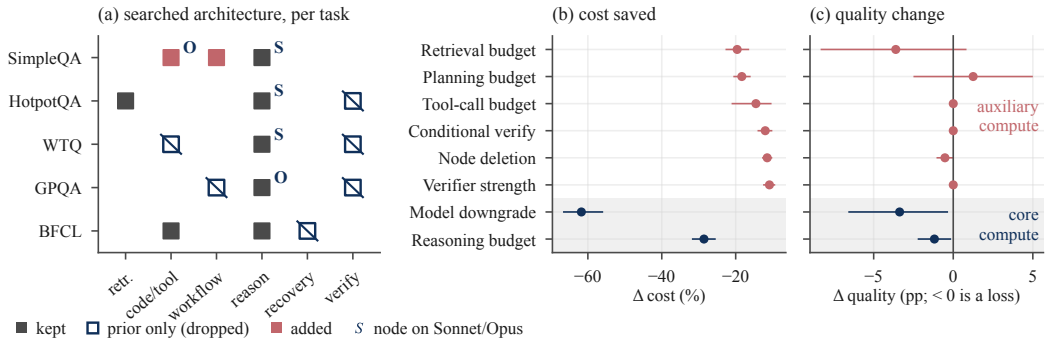


Figure 3: **Search tailors the harness to each task and saves on auxiliary computation at no quality cost.** (a) Components prescribed by the pre-registered prior and kept by search; letters mark nodes moved from Haiku 4.5 to Sonnet (S) or Opus (O). (b, c) Cost and quality change of each cost-reducing operator where search applied it (95% CIs; not additive); a negative quality change is a loss. Shaded: core model computation. `dev`, *in sample*; exact values in appendix D.

5.2 Offline Design and Online Control: Why the Searched Harness Works

We next fix the model and examine what HARP changes in the harness (figure 3; in-sample `dev` unless stated). The searched architectures differ across benchmarks, and so do the nodes that receive a stronger model. Agreement with the task-based prior is 46.7% (figure 3a): search drops every verifier the prior prescribes, and 2 benchmarks reduce to a single reasoning node, because on these tasks a verifier's cost is not repaid in quality.

SimpleQA is the one benchmark where search adds components. For a closed-book task this is unexpected, but the added code node runs on Opus 4.5 and the reasoning node is upgraded to Sonnet. Search uses per-node model assignment to buy knowledge on the nodes that need it, as the knowledge bottleneck of section 5.1 predicts. On HotpotQA, WTQ and GPQA only the reasoning node is upgraded, and on BFCL no node is.

Most of the remaining savings come from auxiliary computation (figure 3b,c). Lowering the tool-call budget saves 14.5% of cost, conditional verification 12.0%, and a cheaper verifier 10.9%, all with no observed quality change. Cuts to core computation move quality: a smaller reasoning budget changes it by -1.2 pp and a node-model downgrade by -3.4 pp, neither significant after Holm correction. At a given price λ, auxiliary budgets are therefore the cheapest quality-neutral savings, which is where the Lagrangian search spends its effort.

One of these cuts shows why held-out validation is still needed. On the BFCL development cell, trace-back removes a further 11.8% of cost with identical scores on all 70 tasks by lowering the tool-call cap (figure A1). On `test`, the same cap solves 0 of 25 multi-turn tasks against 18 for the official harness, since the development split contained few multi-turn tasks.

Removing unneeded computation also explains why online control adds little. HARP-Adaptive allocates computation per episode on top of the searched harness, mapping 25 features to 7 actions. At the 6 operating points where its cost can be matched to the static harness, it ties the static harness at every point, with a detectable effect of ± 9.0–15.0 pp. Once search has removed the computation a task does not need, little remains for a runtime controller to reallocate, and the searched static harness already captures the available gain.

5.3 Generalization of the Searched Harness

A searched harness generalizes across underlying models and usually improves on human-designed harnesses, but it is specific to the task it was searched on (figure 4).

The HotpotQA harness was searched with Haiku 4.5, yet it improves on the default harness under all 4 models after Holm correction (figure 4a). Under the searched harness the four models end at

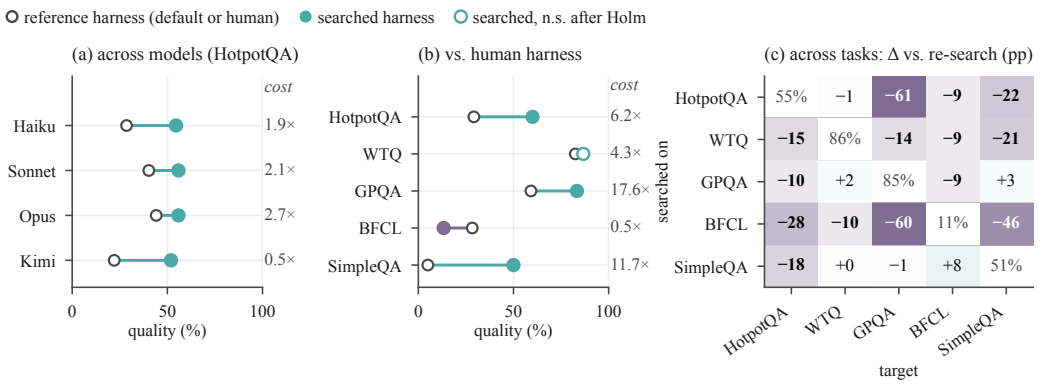


Figure 4: **The searched harness generalizes across models and beats most human-designed harnesses, but is task-specific.** (a) Default → searched harness on HotpotQA, per model. (b) Human-designed → searched harness, per benchmark. Right column: cost ratio. (c) Harness searched on the row benchmark and run on the column benchmark, minus re-searching there (diagonal: re-searched quality); bold: 95% CI excludes zero. `test`; exact values in appendix D.

similar quality, even though their default-harness scores differ widely. The structure search found therefore encodes the task's procedure and carries over to other models.

Against each benchmark's own human-designed harness, the searched harness improves quality by 17.8 pp [14.7, 21.3] pooled over 5 benchmarks, with gains on four of five (three after Holm correction; figure 4b). These gains come at 4.9× the cost, the trade the Lagrangian prices explicitly. WTQ starts near its ceiling and gains little; BFCL is the one loss (-15.0 pp), located entirely in the multi-turn tasks limited by the tool-call cap (section 5.2).

Across tasks, the picture reverses. A harness searched on one benchmark and run on another changes quality by -16.1 pp [-19.4, -12.8] relative to re-searching on the target (figure 4c). The transferred harness carries the source task's structure and budgets, which fit the target poorly, so search has to be run once per task, while the result can be reused across models.

6 Conclusion

Priced side by side, the harness route is always cheaper than the model route: on HotpotQA, Haiku 4.5 with a searched harness beats Opus 4.5 by $+12.5$ pp at 0.04× its cost, and on WTQ and GPQA it buys more quality per dollar than the model upgrade. The method works because search removes computation a task does not need, trims auxiliary budgets at no quality cost, and places stronger models only on the nodes that need them. The resulting structure transfers across models and beats most human-designed harnesses, while each new task needs its own search. The harness route stops paying off when the bottleneck is knowledge, as on SimpleQA, or when a budget cut that is free in sample removes interaction the held-out tasks need, as on BFCL.

AI use statement

We used Claude Code for code implementation, paper drafting, and polishing. The authors reviewed all outputs and take full responsibility for the paper.

Reproducibility statement

The supplement contains the harness runtime, search, analysis and figure generators. Frozen `dev`/`test` splits and pre-registered protocols are described in appendix C, and all costs use one pinned price table with the full run ledger (appendix C). Every reported number is generated from stored run logs.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2507.19457.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *arXiv preprint arXiv:2305.05176*, 2023. URL https://arxiv.org/abs/2305.05176.

Confident AI. DeepEval: The Open-Source LLM Evaluation Framework. Software repository, 2026. URL https://github.com/confident-ai/deepeval. Accessed 2026-09-13.

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. Learning How Hard to Think: Input-Adaptive Allocation of LM Computation. *arXiv preprint arXiv:2410.04707*, 2024. URL https://arxiv.org/abs/2410.04707.

Mehmet Hamza Erol, Batu El, Mirac Suzgun, Mert Yuksekgonul, and James Zou. Cost-of-Pass: An Economic Framework for Evaluating Language Models. *arXiv preprint arXiv:2504.13359*, 2025. URL https://arxiv.org/abs/2504.13359.

Hugh Everett. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research*, 11(3):399–417, 1963.

Shengran Hu, Cong Lu, and Jeff Clune. Automated Design of Agentic Systems. *arXiv preprint arXiv:2408.08435*, 2024. URL https://arxiv.org/abs/2408.08435.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2002.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. *arXiv preprint arXiv:2310.03714*, 2023. URL https://arxiv.org/abs/2310.03714.

LangChain. deepagents: An Open-Source, Batteries-Included Agent Harness. Software repository, 2026. URL https://github.com/langchain-ai/deepagents. Accessed 2026-09-13.

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*, 2026a. URL https://arxiv.org/abs/2603.28052.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: Agent Scaffold for Terminal-Bench 2.0. Software artifact, 2026b. URL https://github.com/stanford-iris-lab/meta-harness-tbench2-artifact. Discovered scaffold artifact.

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. URL https://arxiv.org/abs/2603.03329.

Jessica McFadyen, Ole Jorgensen, Harry Coppock, Kevin Wei, and Cozmin Ududec. How Inference Compute Shapes Frontier LLM Evaluation. *arXiv preprint arXiv:2606.17930*, 2026. URL https://arxiv.org/abs/2606.17930.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, et al. Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks in Command Line Interfaces. *arXiv preprint arXiv:2601.11868*, 2026. URL https://arxiv.org/abs/2601.11868.

Joel Niklaus. Don't Train the Model, Evolve the Harness. Hugging Face Space (research article), 2026a. URL https://huggingface.co/spaces/joelniklaus/harness-optimization. Accessed 2026-09-13.

Joel Niklaus. harness-optimization: Meta-Harness on the Legal Agent Benchmark and a Ten-Harness SWE-bench Pro Comparison. Software repository, 2026b. URL https://github.com/JoelNiklaus/harness-optimization. Accessed 2026-09-13.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to Route LLMs with Preference Data. *arXiv preprint arXiv:2406.18665*, 2024. URL https://arxiv.org/abs/2406.18665.

Jon Saad-Falcon, Adrian Gamarra Lafuente, Shlok Natarajan, Nahum Maru, Hristo Todorov, Etash Guha, E. Kelly Buchanan, Mayee Chen, Neel Guha, Christopher Ré, and Azalia Mirhoseini. Archon: An Architecture Search Framework for Inference-Time Techniques. In *International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2409.15254.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. AgentSquare: Automatic LLM Agent Search in Modular Design Space. *arXiv preprint arXiv:2410.06153*, 2024. URL https://arxiv.org/abs/2410.06153.

Nishad Singhi, Hritik Bansal, Arian Hosseini, Aditya Grover, Kai-Wei Chang, Marcus Rohrbach, and Anna Rohrbach. When To Solve, When To Verify: Compute-Optimal Problem Solving and Generative Verification for LLM Reasoning. In *Conference on Language Modeling (COLM)*, 2025. URL https://arxiv.org/abs/2504.01005.

Prabhakant Sinha and Andris A. Zoltners. The multiple-choice knapsack problem. *Operations Research*, 27(3):503–515, 1979.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. *arXiv preprint arXiv:2408.03314*, 2024. URL https://arxiv.org/abs/2408.03314.

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, Yi Dong, Evelina Bakhturina, Tao Yu, Yejin Choi, Jan Kautz, and Pavlo Molchanov. ToolOrchestra: Elevating Intelligence via Efficient Model and Tool Orchestration. *arXiv preprint arXiv:2511.21689*, 2025. URL https://arxiv.org/abs/2511.21689.

Xinyu Tang, Han Peng, Guoxin Chen, Yuze Shi, Zitao Su, Peiyu Liu, Wayne Xin Zhao, Yawen Li, and Zhe Xue. Agent Systems with Harness Engineering. Survey and curated reading list, 2026. URL https://openreview.net/pdf?id=nM5tDHrQsx. Reading list: https://github.com/RUCAIBox/awesome-agent-harness.

Vivek Trivedy. Better Harness: A Recipe for Harness Hill-Climbing with Evals. LangChain blog, April 2026. URL https://www.langchain.com/blog/better-harness-a-recipe-for-harness-hill-climbing-with-evals. Accessed 2026-09-13.

Laurence A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. *arXiv preprint arXiv:2405.15793*, 2024. URL https://arxiv.org/abs/2405.15793.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating Agentic Workflow Generation. *arXiv preprint arXiv:2410.10762*, 2024. URL https://arxiv.org/abs/2410.10762.

Minhui Zhu, Minyang Tian, Xiaocheng Yang, Tianci Zhou, Penghao Zhu, Eli Chertkov, Shengyan Liu, Yufeng Du, Lifan Yuan, Ziming Ji, Indranil Das, Junyi Cao, et al. Probing the Critical Point (CritPt) of AI Reasoning: a Frontier Physics Research Benchmark. *arXiv preprint arXiv:2509.26574*, 2025. URL https://arxiv.org/abs/2509.26574.

A PROOFS

Proposition 1. (i) If some h' had $Q(h') \geq Q(h_\lambda)$ and $C(h') < C(h_\lambda)$, then $J_\lambda(h') > J_\lambda(h_\lambda)$, contradicting optimality. (ii) Write Q_j, C_j for the values at h_{λ_j}. Optimality gives $Q_1 - \lambda_1 C_1 \geq Q_2 - \lambda_1 C_2$ and $Q_2 - \lambda_2 C_2 \geq Q_1 - \lambda_2 C_1$. Summing yields $(\lambda_2 - \lambda_1)(C_1 - C_2) \geq 0$, so $C_1 \geq C_2$. The first inequality then gives $Q_1 - Q_2 \geq \lambda_1(C_1 - C_2) \geq 0$. (iii) Let $J^\star = J_\lambda(h_\lambda)$. Any h with $Q(h) \geq \tau$ satisfies $C(h) \geq (Q(h) - J^\star)/\lambda \geq (\tau - J^\star)/\lambda$, so the line $C = (\tau - J^\star)/\lambda$ supports $C_m^\star$ from below and touches it at $Q(h_\lambda)$ by (i). Hence the point lies on $\breve{C}_m$. Conversely, each vertex of $\breve{C}_m$ has a supporting line with positive slope $1/\lambda$, and a maximizer of J_λ attains it. For $\tau \in (Q_1, Q_2]$, h_2 is feasible, and $C_m^\star(\tau) \geq \breve{C}_m(\tau) \geq \breve{C}_m(Q_1) = C_1$, because $\breve{C}_m$ is nondecreasing. The mixture has quality τ and cost $pC_2 + (1-p)C_1 = \breve{C}_m(\tau)$ by linearity of $\breve{C}_m$ on $[Q_1, Q_2]$.

Proposition 2. Let $S^\dagger$ attain $C_\tau^\dagger$ and let $d_t = \tau - Q(S_t)$. By monotonicity, $Q(S_t \cup S^\dagger) \geq \tau$, and by the definition of γ, $\sum_{i \in S^\dagger} \Delta_i(S_t) \geq \gamma\, d_t$. Hence some $i \in S^\dagger$ has $\Delta_i(S_t)/c_i \geq \gamma d_t / C_\tau^\dagger$, and the greedy choice satisfies $d_{t+1} \leq d_t\big(1 - \gamma c_{i_t}/C_\tau^\dagger\big) \leq d_t \exp(-\gamma c_{i_t}/C_\tau^\dagger)$. Iterating gives $d_t \leq d_0 \exp(-\gamma \kappa_t / C_\tau^\dagger)$, where κ_t is the cumulative cost. At step $T-1$, $d_{T-1} > \epsilon$, so $\kappa_{T-1} < (C_\tau^\dagger/\gamma)\ln(d_0/\epsilon)$. The final step adds at most $c_{\max}$. For the stopping rule, adding i changes J_λ by $\Delta_i(S) - \lambda c_i < 0$ whenever $\Delta_i(S)/c_i < \lambda$.

Lemma 1. Since $C \geq 0$ and $Q \leq 1$, $J_\lambda \leq 1$. Each promotion raises J_λ by at least ρ on the confidence event, so the number of promotions is at most $(1 - J_\lambda(h_1))/\rho$. At termination, no edit has a certified gain of ρ.

Proposition 3. (i) This is Everett's theorem (Everett, 1963): for any feasible y with cost at most that of $y(\lambda)$, $\widetilde{G}(y) \leq \widetilde{G}(y(\lambda)) + \lambda\big(\mathrm{cost}(y) - \mathrm{cost}(y(\lambda))\big) \leq \widetilde{G}(y(\lambda))$. (ii) The LP relaxation of equation 7 is solved by the per-node upper convex hulls at the critical price. Absent ties, the optimum is integral except at one node, whose fractional part contributes at most $\max_{v,k} \Delta q_{vk}$ (Sinha & Zoltners, 1979), so $U_{\mathrm{frac}} - \widetilde{G}(\hat{y}) \leq \max_{v,k} \Delta q_{vk}$. Then $G(y^\star) \leq \widetilde{G}(y^\star) + \eta \leq U_{\mathrm{frac}} + \eta$ and $G(\hat{y}) \geq \widetilde{G}(\hat{y}) - \eta$.

Proposition 4. $\mathcal{H}$ is fixed before $\mathcal{X}_{\mathrm{cert}}$ is used. One-sided Hoeffding gives $\Pr[\widehat{Q}(h) - Q(h) \geq \beta_n] \leq \delta/(2K)$ and $\Pr[C(h) - \widehat{C}(h) \geq \bar{c}\beta_n] \leq \delta/(2K)$ for each $h \in \mathcal{H}$. A union bound over the $2K$ events completes the proof.

Proposition 5. (i) By the finite-horizon performance difference lemma (Kakade & Langford, 2002), $J_\lambda(\pi^\star_{\mathcal{D}}) - J_\lambda(\hat{\pi}) = \sum_{t=1}^{H} \mathbb{E}_{s_t \sim \hat{\pi}}\big[Q^\star_{\lambda,t}(s_t, a^\star_t) - Q^\star_{\lambda,t}(s_t, \hat{a}_t)\big]$. Since $\hat{a}_t$ maximizes $\widehat{Q}_\lambda$ over $\mathcal{A}_{\mathcal{D}} \ni a^\star_t$, each term is at most $\big(\widehat{Q}_\lambda(s_t, a^\star_t) + \xi\big) - \big(\widehat{Q}_\lambda(s_t, \hat{a}_t) - \xi\big) \leq 2\xi$. (ii) Apply (i) and subtract the priced observation cost $\lambda H c_{\mathrm{obs}}$. $\Delta_{\mathrm{adapt}} \geq 0$ because $\pi^{\mathrm{st}} \in \Pi_{\mathcal{D}}$. (iii) If every executed action satisfies $c(a_t) \leq \kappa \bar{c}_{a_t} \leq b_t$, the remaining budget never becomes negative. A union bound over at most H steps gives the claim.

B METHOD DETAILS

Algorithm 1 summarizes HARP. The remaining implementation details are as follows.

Algorithm 1 HARP: primal–dual harness optimization and support-constrained control

Input: Dev. tasks $\mathcal{X}_{\text{dev}}$, certification tasks $\mathcal{X}_{\text{cert}}$, rich harness h_R, components $\mathcal{C}$, target τ, budget B, tolerances ϵ, ρ, δ
Output: Certified harness $h^\star$, price $\lambda^\star$, controller $\hat{\pi}$

Run h_R and h_R^{-i}, $i \in \mathcal{C}$, on $\mathcal{X}_{\text{dev}}$; initialize trace set $\mathcal{D}$ and $\mathcal{H} \leftarrow \emptyset$
for each price λ placed by bisection on $\log \lambda$ **do** ▷ Prop. 1
  // Stage 1: composition (Prop. 2, Rem. 1)
  $S^{\rightarrow} \leftarrow$ ratio greedy equation 4, stopped when $\max_i \text{LCB}[\Delta_i]/c_i < \lambda$
  $S^{\leftarrow} \leftarrow$ longest prefix of $\text{argsort}(\ell)$ removable with LCB drop $\leq \epsilon$
  $h \leftarrow \arg\max\{J_\lambda(S^{\rightarrow}),\ J_\lambda(S^{\leftarrow})\}$
  // Stage 2: topology (Lemma 1)
  while some $o \in \mathcal{O}(h)$ satisfies equation 6 **do**
    $h \leftarrow o(h)$ for the best such o
  end while
  // Stage 3: configuration and model assignment (Prop. 3)
  $k_v \leftarrow \arg\max_k\ \Delta q_{vk} - \lambda\, \Delta c_{vk}$ for every node v; $h_\lambda \leftarrow h$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{h_\lambda\}$; add all evaluated traces to $\mathcal{D}$
end for
// Certification on held-out tasks (Prop. 4)
$h^\star \leftarrow$ solution of equation 10 on $\mathcal{X}_{\text{cert}}$; $\lambda^\star \leftarrow$ its price
// Online control (Prop. 5)
Fit $\widehat{Q}_{\lambda^\star}$ on $\mathcal{D}$; deploy $\hat{\pi}(s, b) = \arg\max_{a \in \mathcal{A}_{\mathcal{D}}(s,b)} \widehat{Q}_{\lambda^\star}(s, a)$

- **Executors.** Each node kind (retrieval, code/tool, workflow, reasoning, verifier, recovery) declares what it reads and writes, whether it calls an LLM, its configuration range, and the models it may be assigned.
- **Guards.** A guard branches on a projection of the execution state (`confidence`, `verdict`, `attempts`, `usd`, `steps`); back-edges are bounded, so the forward graph stays acyclic.
- **Search space.** Nine deletable node kinds, the configuration grids and per-node model assignment give a joint space of order 10^{33}; coarsened grids and role-restricted assignment reduce the searched support to order 10^8, of which search visits order 10^3 specifications.
- **Operators.** Gate or relocate a verifier, short-circuit, memoize retrieval, merge duplicated computation, substitute a cheaper tool, and demote a node's model.
- **Promotion.** Candidates run in dollar-budgeted rungs and are promoted on J_λ, cheapest first.
- **Budget matching for online control.** λ is calibrated against the budget net of probing cost; the static comparator is placed at the controller's realized cost by mixing the two bracketing static harnesses; probe calls are charged to the adaptive arm.

C Experimental protocol

Table A1: **Benchmarks.** Split sizes are frozen before any run.

Benchmark	Task demand	Metric	`dev` / `test`
SimpleQA	closed-book factual QA	rubric judge	80 / 150
HotpotQA	multi-hop retrieval	exact match	72 / 144
WikiTableQuestions	structured calculation	denotation accuracy	70 / 140
GPQA-Diamond	difficult reasoning	choice accuracy	66 / 132
BFCL	tool-use workflow	AST / state checkers	70 / 140
τ-bench	long-horizon workflow	environment reward	56 / 109

Splits and grading. Search and all in-sample analyses use `dev`; every headline comparison uses the held-out `test` split. Graders are offline re-implementations of each benchmark's official checker (a rubric judge for SimpleQA). Degradation and failure injectors are seeded by (task, condition, seed).

Statistics. Every interval is a paired cluster bootstrap over task ids (10 000 resamples), BCa where valid and percentile otherwise. p values are bootstrap achieved significance levels; Holm correction is applied within each declared family of comparisons. Nulls are reported with the smallest effect they could have detected.

Pre-registration. Each experiment block has a hypothesis, decision rule and falsifier registered before its run (`specs/preregistration/`, blocks `B01`–`B08`). Deviations: headline blocks ran one seed per task and arm instead of three; one spend cap was amended in writing before the runs it governs (`A-2026-09-14-1`), which leaves τ-bench with a model axis only; one late block has no registration of its own.

Exclusions. Timeouts and budget exhaustions are dropped by one rule applied everywhere, and reduced n is printed. Cells below the coverage floor, and slices measured on a harness superseded by a later fix, are excluded; their data remain on disk.

Cost accounting. All dollars come from one pinned price table (price_table@058426112e6a). We report *priced* cost: a cached call is charged at list price, so the study's priced total ($468.91) exceeds the amount billed, and search cost $c_{\text{opt}} = \$219.01$ corresponds to $49.04 of gateway spend. Break-even estimates are therefore conservative. c_{opt} accumulates over all search stages that produced a harness.

D SUPPLEMENTARY RESULTS

Exact values behind figure 2, figure 3 and figure 4.

D.1 MODEL CAPABILITY VERSUS HARNESS CAPABILITY

Table A2: **HARP-Haiku versus Opus with the default harness** (figure 2). Δ: HARP-Haiku minus Opus, 95% BCa CI; cost: HARP-Haiku as a multiple of Opus. Bold: significant after Holm. `test`.

Benchmark	Haiku	HARP-Haiku	Opus	Δ (pp) [95% CI]	Cost
HotpotQA	31.3	56.3	43.8	**+12.5 [+6.3, +19.4]**	0.04×
WTQ	82.1	84.3	87.9	−3.6 [−7.9, −0.7]	0.34×
GPQA	59.7	70.5	79.1	−8.5 [−16.3, −0.8]	0.59×
BFCL	28.1	15.1	30.2	**−15.1 [−23.0, −7.9]**	0.10×
SimpleQA	6.8	6.8	40.5	**−33.8 [−41.9, −25.7]**	0.19×

Only one matched-quality comparison is identified within measured support (SimpleQA, optimized versus default harness on the expensive model); its break-even requires 1.72×10^6 served tasks and the underlying saving is not significant. All other iso-cost and iso-quality cells fall outside measured support and are not reported.

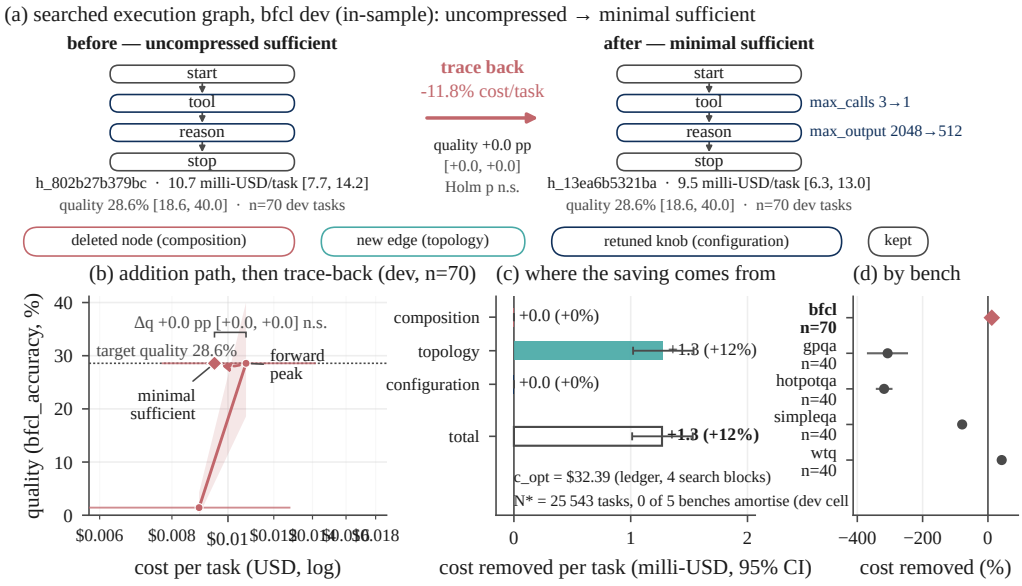
D.2 OFFLINE HARNESS DESIGN

Table A3: **Searched architecture per benchmark** (figure 3a). Families: ret(rieval), code, w(ork)f(low), r(ea)s(o)n, v(eri)f(ier), rec(overy). `dev`, *in sample*; quality with 95% CI.

Benchmark	Prior	Searched	Node models (off Haiku)	Quality (%)	USD/task
SimpleQA	rsn	code+wf+rsn	code→Opus, reason→Sonnet	55.0 (40.0–70.0)	0.0087
HotpotQA	ret+rsn+vfy	ret+rsn	reason→Sonnet	65.0 (50.0–80.0)	0.0056
WTQ	code+rsn+vfy	rsn	reason→Sonnet	90.0 (80.0–97.5)	0.0076
GPQA	wf+rsn+vfy	rsn	reason→Opus	77.5 (62.5–90.0)	0.0740
BFCL	code+rsn+rec	code+rsn	—	28.6 (18.6–40.0)	0.0095

Table A4: **Cost-reducing operators** (figure 3b,c), measured where search applied them; effects are not additive. p_{Holm} is for the quality change. Five further operators (aggregation, cheap tool, early exit, retrieval dedup, retry budget) are exact no-ops. `dev`, *in sample*.

Operator	n	Δcost, % (95% CI)	Δquality, pp (95% CI)	p_{Holm}	Search
Auxiliary computation					
Retrieval budget	72	−19.6 (−22.8, −16.3)	−3.6 (−8.3, +0.8)	1.00	mixed
Planning budget	80	−18.3 (−20.6, −15.9)	+1.2 (−2.5, +5.0)	1.00	kept
Tool-call budget	70	−14.5 (−21.1, −10.3)	0.0 (0.0, 0.0)	1.00	applied
Conditional verify	358	−12.0 (−14.1, −10.1)	0.0 (0.0, 0.0)	1.00	declined
Node deletion	358	−11.5 (−12.8, −10.1)	−0.5 (−1.1, 0.0)	0.63	mixed
Verifier strength	358	−10.9 (−12.6, −9.3)	0.0 (0.0, 0.0)	1.00	declined
Core computation					
Model downgrade	230	−61.8 (−66.7, −55.9)	−3.4 (−6.6, −0.3)	0.39	kept
Reasoning budget	358	−28.6 (−31.8, −25.4)	−1.2 (−2.2, −0.1)	0.39	applied



split: dev, in-sample (B02/B03/B04). attribution: telescoping along the search's own stage order, levels sum to the total exactly. Paired 10k bootstrap 95% CIs; metric bfcl_accuracy; usd_priced (price_table@058426112e6a). AMORTISATION: N* = c_opt/Δc_inf exists only on this dev cell; on the campaign break-even basis (best arm vs modal baseline) 0 of 5 benches amortise, bfcl −$0.0407/episode, N* undefined.

Figure A1: **Trace-back on the BFCL development cell removes 11.8% of cost by retuning budgets.** Lowering the tool-call cap and the output length leaves all 70 paired tasks unchanged ($p_{\text{Holm}} = 1.000$); the same cap underlies the held-out multi-turn loss (appendix D.3). `dev`, *in sample*.

D.3 TRANSFER

Table A5: **Transfer across tasks, models and harness designers** (figure 4). Cost ratio: searched over reference; search cost is not charged. *Not significant after Holm. `test`.

	Δquality, pp (95% CI)	Cost ratio (95% CI)
A. Across tasks: transferred harness minus re-search on the target (Haiku)		
→ SimpleQA	−21.9 (−30.6, −13.8)	0.44 (0.42, 0.45)
→ HotpotQA	−17.5 (−25.9, −11.3)	1.04 (0.95, 1.17)
→ WTQ	−2.2 (−5.9, +0.6)	1.11 (1.06, 1.18)
→ GPQA	−34.1 (−41.2, −24.7)	0.20 (0.18, 0.22)
→ BFCL	−4.7 (−13.8, +3.1)	1.72 (1.53, 1.99)
B. Across models: searched minus default harness, HotpotQA		
Haiku	+26.0 (+15.6, +37.7)	1.91 (1.67, 2.16)
Sonnet	+15.6 (+7.8, +24.7)	2.11 (1.89, 2.32)
Opus	+11.7 (+5.2, +20.8)	2.70 (2.39, 2.93)
Kimi	+29.9 (+19.5, +41.6)	0.55 (0.40, 0.81)
Haiku + searched vs. Opus + default	+10.4 (+3.9, +19.5)	0.55 (0.49, 0.60)
C. Against human-designed harnesses (human-harness quality in brackets)		
SimpleQA [5.0%]	+45.0 (+36.7, +54.2)	11.70 (10.99, 12.44)
HotpotQA [29.2%]	+30.8 (+22.5, +39.2)	6.16 (5.57, 6.79)
WTQ [82.5%]	+4.2 (+0.8, +10.0)*	4.35 (4.11, 4.56)
GPQA [59.2%]	+24.2 (+16.7, +33.3)	17.61 (16.46, 18.88)
BFCL [28.3%]	−15.0 (−21.7, −9.2)	0.51 (0.45, 0.57)
Pooled	+17.8 (+14.7, +21.3)	4.90 (4.71, 5.08)

Why the BFCL harness fails. The searched harness keeps its tool node (`start→tool→reason→stop`), and the loss comes from its budget. Search lowered the tool-call cap below the official harness's cap, which cost nothing on `dev`. On `test`, single-turn tasks tie exactly, while the searched harness solves 0 of 25 multi-turn tasks (official: 18).