DYNAMIC HARNESS SEARCH: BUILDING MULTI-AGENT SYSTEMS PER-QUERY VIA PREDICTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent harnesses specify the roles, instructions, tools, and communication structure used to solve a task, and the right harness depends on the query. Because the value of each design choice is observable only through execution, tailoring a harness to each query has required either executing alternatives at inference time or costly manual design. We introduce SHIFT, which moves execution out of the per-query search loop. A local LLM architect learns a policy over harness-building actions from search, and a value function that predicts, from measured executions, a utility balancing accuracy against execution cost. For each query, Monte Carlo tree search uses these predictions to construct a harness. Across 9,193 tasks in six benchmarks, from math to document and general-assistant tasks, with a Gemini 3.5 Flash executor, SHIFT attains the highest mean accuracy, about 80%, outperforming 17 baselines that span prompting, prompt optimization, and workflow search, and exceeding the strongest baseline by 7.2 percentage points. A cheaper mode of SHIFT also attains a higher mean accuracy than every baseline while using 32% fewer execution tokens than the strongest baseline. We further show that choosing structure, instructions, and tools jointly beats choosing only instructions or only tools by up to 9.1 percentage points, and that learned value selection identifies more accurate harnesses with lower execution cost from candidate pools.

1 INTRODUCTION

Consider an assistant that handles requests over an organization's documents. One user asks for a number stated in a supplied report; another asks the assistant to reconcile a budget workbook and return a corrected file. The first request may need only a lookup. The second requires reading several sheets, performing calculations, editing cells, and checking the result, work that can be divided among a planner, a solver, and a verifier Yao et al. (2022); Wu et al. (2023). Building such a system means deciding which agents to use, what instructions each follows, which tools it may call, and how agents pass results to one another. We call this specification the *harness*. Yet the planner and verifier that help with the workbook may add unnecessary model calls and latency to the lookup. The right harness therefore depends on the query: it must provide the capabilities the task requires while keeping execution cost low.

Choosing this balance is difficult because harness components interact. A verifier's usefulness depends on the solver's output and the instructions for checking it; access to a tool helps only if the agent uses it effectively. These interactions make a component's value depend on both the query and the rest of the harness. Evaluating alternative designs through execution, however, adds to the cost of answering each query. Existing methods address this expense in different ways: workflow optimization searches offline for a reusable structure Zhang et al. (2025b); Zhuge et al. (2024), while prompt optimization tunes instructions within a fixed program Khattab et al. (2024); Opsahl-Ong et al. (2024). Query-adaptive controllers generate workflows directly Zhang et al. (2025a); Taparia et al. (2026), routers select among predefined models or workflows Chen et al. (2023); Ong et al. (2025); Yue et al. (2025), and learned quality predictors guide offline workflow search Shang et al. (2025). These approaches motivate our question: can a learned estimate of harness utility guide search over interacting design choices for each query while balancing accuracy against execution cost, allowing alternatives to be compared before any is executed?

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.

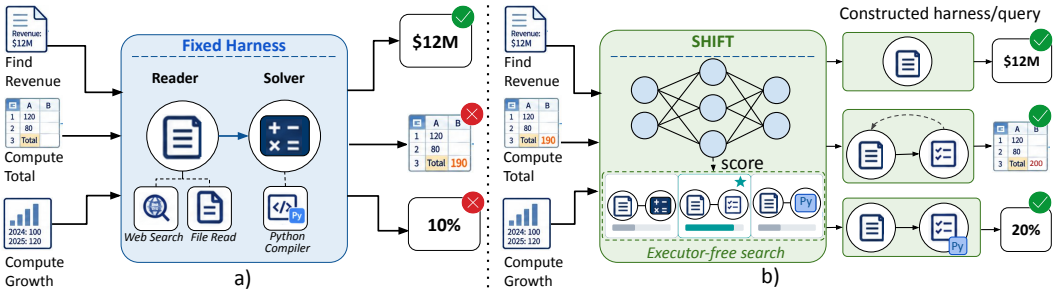

Figure 1: **Tailoring the harness to the query.** Three queries ask for a reported revenue, a column total, and a growth rate. (a) A fixed harness runs the same Reader and Solver, with all their tools, on every query: it spends this structure on a simple lookup and still gets both calculations wrong (190 instead of 200, 10% instead of 20%). (b) SHIFT constructs a harness for each query, for example a single reader for the lookup, a reader with a verifier loop for the total, and a reader and a verifier with Python for the growth rate. The architect scores candidates without executing them, and only the constructed harness is run; in this example, all three answers are correct.

We introduce SHIFT (Searching Harnesses In Forward-pass Trees), a framework that constructs an agent harness for each query without executing candidate designs during search. At its core is a small local model, the *architect*, which proposes harness-building actions and predicts a utility balancing accuracy against execution cost. Monte Carlo tree search uses these predictions to explore combinations of agents, instructions, and tool permissions through architect inference alone (Figure 1). The architect learns from search decisions and measured execution outcomes during training, whose cost is amortized over subsequent queries. At inference time, only the selected harness is executed.

We evaluate SHIFT on six benchmarks spanning math, code, multi-hop and document question answering, spreadsheets, and general assistance, against 17 baselines that share the same executor (Table 1). SHIFT-search selects the harness preferred by search and achieves the highest mean accuracy, 79.9%, exceeding the strongest baseline by 7.2 percentage points. Its largest gains occur on OfficeQA and GAIA, where it improves accuracy over that baseline by 30.0 and 15.0 percentage points, respectively, while remaining within 1.2 percentage points of the best-performing methods on the other four benchmarks. We also introduce a lower-cost variant, SHIFT-value which selects using predicted utility and achieves 74.5% mean accuracy, exceeding every baseline while using 32% fewer execution tokens than the strongest baseline.

Our contributions are as follows:

1. We formulate per-query harness construction as tree search over agent, instruction, and tool actions, guided by a learned architect that predicts the harness utility without executing it.
2. On six benchmarks with a shared executor, SHIFT-search is more accurate than all 17 baselines, and SHIFT-value exceeds them all in mean accuracy at 32% lower token cost.
3. We show that optimizing structure, instructions, and tools jointly beats any one alone, and the learned architect picks accurate, low-cost harnesses and transfers to harder tasks.

2 Related Work

Prior work improves agent systems through reasoning and prompt design, automated workflow optimization, and query-adaptive composition. We review these directions with particular attention to how candidate designs are evaluated and when adaptation occurs.

Reasoning and prompt optimization. ReAct interleaves reasoning with actions so that agents can respond to tool observations Yao et al. (2022). Self-Refine uses iterative feedback to revise outputs Madaan et al. (2023), while Tree of Thoughts and Graph of Thoughts organize intermediate reasoning into branches or graphs that support exploring and comparing possible solutions Yao et al. (2023); Besta et al. (2024). Multi-agent frameworks distribute tasks among agents through role assignments and communication protocols Wu et al. (2023); Hong et al. (2024); Chen et al. (2024). The effectiveness of these reasoning and coordination mechanisms also depends on their instructions. Prompt optimization improves the instructions and few-shot examples of a fixed program using evaluation feedback, as in DSPy, MIPRO, and GEPA Khattab et al. (2024); Opsahl-Ong et al. (2024); Agrawal et al. (2026). SHIFT instead makes the harness itself the object of search be-

fore execution, choosing its agent structure, instructions, and tool permissions jointly as interacting design choices.

Workflow optimization and surrogate evaluation. Automating workflow design requires a way to judge whether a proposed configuration is worth keeping. Execution feedback supplies this signal directly: workflow and program optimizers use measured outcomes to revise executable code, agent graphs, or their constituent instructions Zhang et al. (2025b); Zhuge et al. (2024); Cheng et al. (2024); Hu et al. (2025). When optimization produces a reusable design, its evaluation cost can be spread across subsequent queries. Constructing a new harness for each query makes repeated candidate execution more costly, motivating learned estimates of candidate quality. These estimates serve as surrogates for the measured outcomes of running a candidate. AgentSquare uses an LLM performance predictor to assess module recombinations Shang et al. (2025); however, newly evolved modules are still evaluated by execution in the task environment. SHIFT trains a local utility predictor on executed harnesses and uses its predictions to guide a new tree search for each query.

Query-adaptive composition and routing. The best agent configuration can vary with the query, motivating controllers that allocate computation or compose workflows per query. Model routers choose among language models Chen et al. (2023); Ong et al. (2025), while controllers for multi-agent systems also select collaboration modes, roles, and model assignments Yue et al. (2025). Adaptation can extend to the workflow itself: MaAS samples query-conditioned operator compositions from a supernet Zhang et al. (2025a). Other approaches generate communication topologies or executable multi-agent systems Li et al. (2026); Ye et al. (2025), or configure workflows together with tools, token budgets, and prompts Taparia et al. (2026). SHIFT combines query-specific construction with explicit lookahead over agent, instruction, and tool choices. Lookahead considers how an action affects the harness built by subsequent actions. A learned policy proposes actions, while a utility predictor trained on execution outcomes evaluates candidate harnesses during Monte Carlo tree search. This allows SHIFT to compare alternative designs for each query without executing candidates during inference-time search.

3 METHODOLOGY

SHIFT builds a harness for each query by incrementally adding agents, instructions, and tools to an executable graph. Every intermediate graph is runnable, allowing search to compare designs at different construction stages (Section 3.1). A small local model, the *architect*, proposes actions and predicts harness utility (Section 3.2). These predictions allow Monte Carlo tree search to explore how successive choices work together without executing candidate designs; only the selected harness is run (Section 3.3). The architect learns these predictions from search decisions and measured execution outcomes during training (Section 3.4). Figure 3 illustrates the framework, and Algorithm 1 details the search.

3.1 HARNESSES AS EXECUTABLE GRAPHS

We represent a harness as a directed graph, because a multi-agent harness is naturally described by who does what and who hands results to whom. Formally, a harness state s is a graph $G = (\mathcal{V}, \mathcal{E})$ with a start node $v_0 \in \mathcal{V}$. Each agent $v \in \mathcal{V}$ is specified by a role ρ_v, an ordered list of directives D_v, and a set of permitted tools $T_v \subseteq \mathcal{K}_d$, where $\mathcal{K}_d$ is the tool set available in domain d. An edge $(u, v) \in \mathcal{E}$ passes the output of u to v. Feedback edges let downstream agents return results for revision. Executing G on a query q, starting from v_0, produces an answer through the agents' reasoning and tool calls Yao et al. (2022). This representation has two advantages. First, it places a single reasoning agent, a retrieval-and-synthesis pipeline, and a workflow with verification loops in one space, so one

	Agents 1 → 4+	Tools none → all	Directives 0 → 2+
GSM8K	**−0.09** 0% acc · 19× tok	**−0.09** −1% acc · 17× tok	**−0.03** 0% acc · 3× tok
MBPP	**−0.02** +5% acc · 3× tok	**−0.02** +4% acc · 8× tok	**−0.02** +2% acc · 3× tok
HotpotQA	**−0.03** +9% acc · 13× tok	**−0.02** +11% acc · 15× tok	**−0.02** +3% acc · 2× tok
Spreadsheet-Bench	**+0.13** +28% acc · 14× tok	**+0.24** +42% acc · 30× tok	**+0.10** +18% acc · 2× tok
GAIA	**+0.17** +34% acc · 10× tok	**+0.14** +32% acc · 61× tok	**+0.13** +22% acc · 2× tok
OfficeQA	**+0.27** +50% acc · 22× tok	**+0.14** +31% acc · 212× tok	**+0.15** +28% acc · 4× tok

Figure 2: **No single harness fits every task.** Utility change (success minus cost) between harnesses with varying structure; small text: accuracy change and token multiplier. More structure can accompany higher accuracy yet lower utility.

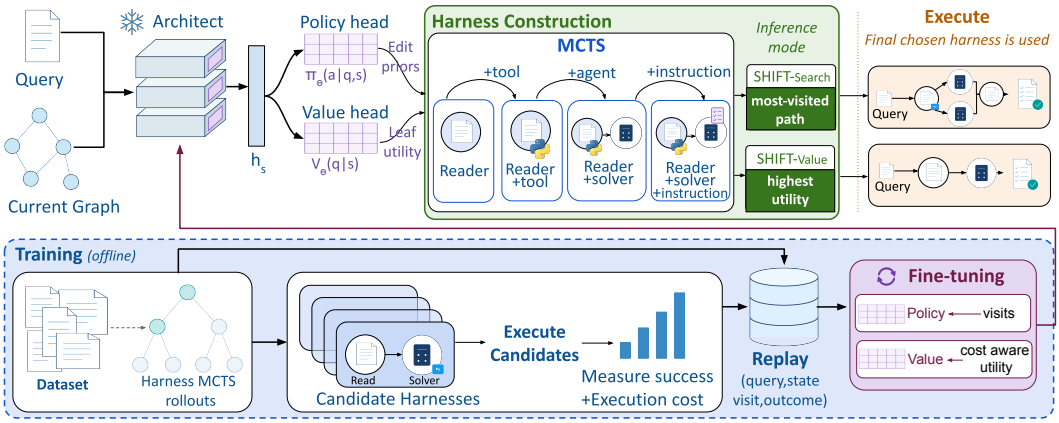

Figure 3: **Overview of SHIFT.** Given a query, the current harness, and its feasible actions, a local Gemma-2-2B architect produces a policy over actions and a utility estimate. MCTS uses these predictions to search over action sequences without executor calls, after which the executor runs only the selected harness. During training, measured outcomes are stored in a replay buffer and used to update the LoRA adapters and policy–value heads.

search procedure covers all of them. Second, it supports explicit checks on graph structure and tool permissions, so search can exclude invalid harnesses.

A harness-building *action* modifies structure, instructions, or tool permissions: it inserts an agent with its edges, adds a feedback edge, appends a directive to some D_v, or expands one or more tool sets T_v. Adding a Planner, for example, introduces a planning stage, and granting a Reasoner a file reader gives it access to supporting documents. Their benefit varies sharply across tasks (Figure 2). On financial-document questions, harnesses with four or more agents have about 50 percentage points higher accuracy than single-agent harnesses; on arithmetic word problems, they use about 19 times as many execution tokens with little accuracy difference. Components come from a user-defined library, and SHIFT decides for each query how to compose them. Full specifications of role instructions, the initial harness, the action library, and the search settings are given in Appendix A.

Algorithm 1: Per-query harness construction. The complete procedure is Algorithm 2 (Appendix A).

Require: query q, architect (π_θ, V_θ), budget B
1: $\mathcal{T} \leftarrow$ tree rooted at seed harness s_0
2: **for** $b = 1, \ldots, B$ **do**
3: descend $\mathcal{T}$ by Eq. (3) to leaf s
4: expand s with priors from π_θ
5: back up $V_\theta(q, s)$ ▷ no executor
6: **end for**
7: SEARCH: $s^\star \leftarrow$ most-visited path's end
8: VALUE: $s^\star \leftarrow \arg\max V_\theta$ over candidates
9: **return** EXECUTE$(s^\star, q)$ ▷ one call

A sequence of at most H actions defines a finite-horizon decision process over harness states. Let $\mathcal{S}_d$ be the set of executable harnesses that use only tools in $\mathcal{K}_d$, and $\mathcal{A}_d$ the action vocabulary. Each action a_t transforms s_t into $s_{t+1} = M(s_t, a_t)$ under a deterministic transition M. Feasible actions either modify the harness while preserving executability or stop construction:

$$\mathcal{A}(s) = \{a \in \mathcal{A}_d : M(s,a) \in \mathcal{S}_d,\ M(s,a) \neq s\} \cup \{a_{\text{stop}}\}. \tag{1}$$

From a minimal harness $s_0 \in \mathcal{S}_d$, a single Coder without tools, the architect chooses actions given the query and the current harness. Construction stops at a_{stop} or after H actions; since every state is executable, search can compare partial and complete designs.

3.2 The Policy–Value Architect

A Gemma 4 E2B backbone Gemma Team (2026) reads the query, available tools, feasible actions, and a text rendering of the current graph. Its final-token hidden state is $h_\theta(q, s)$. Two linear heads map it to action logits $\ell_{\theta,a}(q, s)$ and a value logit $\hat{v}_\theta(q, s)$. Only LoRA adapters Hu et al. (2021) and the two heads, collectively θ, are trained; the backbone weights remain frozen. The policy is normalized over feasible actions $\mathcal{A}(s)$; all others receive zero probability:

$$\pi_\theta(a \mid q, s) = \frac{\mathbf{1}[a \in \mathcal{A}(s)] \exp(\ell_{\theta,a}(q,s)/T_\pi)}{\sum_{b \in \mathcal{A}(s)} \exp(\ell_{\theta,b}(q,s)/T_\pi)}, \qquad V_\theta(q,s) = \sigma(\hat{v}_\theta(q,s)). \tag{2}$$

Here $T_\pi > 0$ is the policy temperature and σ is the sigmoid. The policy π_θ guides exploration, while $V_\theta(q, s) \in (0, 1)$ estimates utility, balancing task success against resource use (Section 3.4). Neither prediction requires running the harness.

3.3 Searching over Harnesses

SHIFT searches for a harness with Monte Carlo tree search (MCTS) Silver et al. (2017); Hao et al. (2023), guided by the architect's policy and value. MCTS suits harness design for three reasons. First, the transition M is known and deterministic, so candidate harnesses can be constructed without execution. The architect estimates their utility through local inference, allowing search to compare alternatives before selecting one to run. Second, design choices interact: an agent may become useful only when paired with suitable instructions or tools. MCTS captures these dependencies by evaluating actions through the designs they lead to. Third, search visit counts provide training targets for the policy Silver et al. (2017), allowing the architect to learn from search and guide future searches toward promising designs.

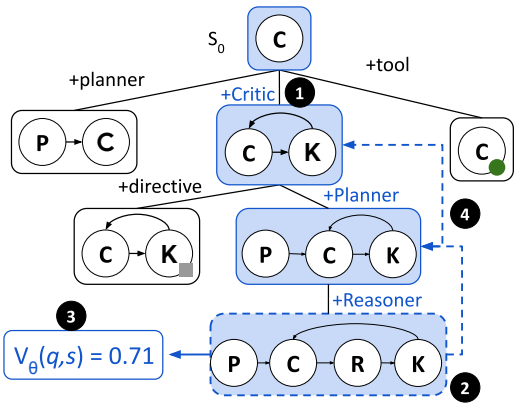

Figure 4: One search simulation: (1) select, (2) expand, (3) score with V_θ (4) back up. C: Coder, P: Planner, R: Reasoner, K: Critic.

The search tree is rooted at s_0, with harness states as nodes and actions as branches. When search first reaches a node, the architect predicts its policy and utility. The K_{exp} highest-probability feasible actions, together with a_{stop}, form the expanded set $U(s)$. The policy is then renormalized over $U(s)$ to obtain the search prior $P(a \mid s)$. Each of the B iterations then descends from the root, choosing at each expanded node:

$$a^\star = \arg\max_{a \in U(s)} \left[\widetilde{Q}(s, a) + c_{\text{puct}} P(a \mid s) \frac{\sqrt{\max(1, \widetilde{N}(s))}}{1 + \widetilde{N}(s, a)} \right], \tag{3}$$

where N counts visits, $Q(s, a)$ is the mean value backed up through action a, and $c_{\text{puct}} > 0$ balances the two terms; tildes mark statistics adjusted for parallel search (Appendix A). The first term favors actions whose continuations have scored well, and the second favors promising actions that have been tried less often. When the descent reaches a new node, selects a_{stop}, or hits depth H, the leaf's value is passed back along the path to update N and Q. The executor is never called inside this loop.

Once the budget is spent, SHIFT-search executes the harness at the end of the most-visited path, and SHIFT-value executes the candidate from the most-visited branches with the highest predicted utility, searching again with a larger budget if that value is low (Appendix B).

Amortized design cost. SHIFT replaces executor calls with local inference. Training is paid once and amortized over later queries, whereas search runs for every query and pays off when it makes execution cheaper: search matches greedy construction in accuracy at about 30% lower executor-token cost (Section 4.3).

3.4 Learning from Execution

The architect learns its policy from search and its value estimates from executed harnesses within a shared training loop (Algorithm 2). For each training query in $\mathcal{Q}_d$, SHIFT runs search with exploration noise at the root, samples a construction path according to visit counts, and executes the resulting harness and up to K_{exec} other candidates from the tree.

Cost-aware reward. Each executed harness receives a reward that credits task success and subtracts penalties for execution tokens, latency, timeouts, and additional tools and agents, each scaled by a domain-specific reference. The reward is clipped and rescaled to a target $z \in [0, 1]$, the utility the value head learns to predict (Equation 7, Appendix A).

Training targets. Each training example i pairs a query q_i with a harness state s_i. If search visited children of s_i, its policy target is the share of visits each expanded action received, $p_i(a) = N(s_i, a)/\sum_{b \in U(s_i)} N(s_i, b)$ for $a \in U(s_i)$. If s_i was executed, its value target is its measured utility z_i. Intermediate states thus never inherit another harness's outcome.

Online objective. Examples are drawn from a replay buffer in minibatches $\mathcal{I}$ that favor informative examples, with importance weights $w_i > 0$ correcting for this bias. Let $\mathcal{I}_\pi \subseteq \mathcal{I}$ and $\mathcal{I}_V \subseteq \mathcal{I}$ be the examples with policy and value targets. The architect minimizes

$$\mathcal{L}_{\text{online}} = \frac{1}{|\mathcal{I}|}\Big[\sum_{i \in \mathcal{I}_\pi} w_i\big(\,\text{CE}(p_i, \pi_i^U) - \beta\,\mathcal{H}(\pi_i)\big) + \lambda_v \sum_{i \in \mathcal{I}_V} w_i\,\text{BCE}(z_i, V_i)\Big]. \tag{4}$$

The first sum trains the policy $\pi_i = \pi_\theta(\cdot \mid q_i, s_i)$ to match the search visit distribution p_i. Cross-entropy is computed using π_i^U, the policy renormalized over expanded actions $U(s_i)$, so unexamined actions do not count as negative evidence. The entropy term $\mathcal{H}(\pi_i)$, weighted by $\beta \geq 0$, encourages exploration. The second sum trains $V_i = V_\theta(q_i, s_i)$ to predict measured utility z_i using binary cross-entropy, weighted by $\lambda_v \geq 0$.

Ranking objective. Search needs to distinguish better harnesses from worse ones for the same query. At the end of each epoch, we therefore add a ranking loss that trains the value head to preserve their measured utility ordering. Let $\mathcal{J}$ be a sample of pairs (i, j) executed on the same query whose utilities differ by at least $\delta > 0$, and let $\hat{v}_i = \hat{v}_\theta(q_i, s_i)$ be the value logit before the sigmoid. Then

$$\mathcal{L}_{\text{rank}} = \frac{1}{|\mathcal{J}|} \sum_{(i,j) \in \mathcal{J}} \max\{0,\; m - \text{sgn}(z_i - z_j)(\hat{v}_i - \hat{v}_j)\}, \tag{5}$$

where $\text{sgn}(z_i - z_j)$ is $+1$ if harness i scored higher and -1 otherwise. The loss is zero once the better harness leads by at least the margin $m > 0$ in logit space and grows linearly otherwise; it is added to the value cross-entropy with weight $\lambda_r \geq 0$.

4 Experimental Results

We evaluate SHIFT on six benchmarks that range from short arithmetic problems to long, tool-heavy document and assistant tasks. Section 4.2 compares the accuracy of SHIFT's harnesses with fixed and learned workflow designs, Section 4.3 examines how search trades accuracy against execution cost, Section 4.4 identifies the components that account for the gains, and Section 4.5 tests transfer to harder tasks without retraining.

4.1 Evaluation Setup

Benchmarks and baselines. We evaluate mathematical reasoning (GSM8K Cobbe et al. (2021)), multi-hop question answering (HotpotQA Yang et al. (2018)), code generation (MBPP Austin et al. (2021)), spreadsheet manipulation (SpreadsheetBench Ma et al. (2024)), document-grounded question answering (OfficeQA Pro Opsahl-Ong et al. (2026)), and general-purpose assistance (GAIA Mialon et al. (2024)), for a total of 9,193 evaluation tasks. We use official splits where they exist and study-specific partitions for SpreadsheetBench, OfficeQA, and GAIA (Appendix C). We compare against 17 baselines in the three groups of Section 2: fixed harnesses, spanning prompting strategies Wei et al. (2022); Wang et al. (2022; 2023); Madaan et al. (2023); Yao et al. (2023), tool-using and multi-agent frameworks Yao et al. (2022); Chen et al. (2024), and DSPy programs with access to every tool, unoptimized or optimized with MIPROv2 and GEPA Khattab et al. (2024); Opsahl-Ong et al. (2024); Agrawal et al. (2026); workflow optimizers Zhang et al. (2025b); Zhuge et al. (2024); Cheng et al. (2024); and query-adaptive composition Liu et al. (2023). All methods use Gemini 3.5 Flash Gemini Team (2023) as the executor unless otherwise specified. Appendix D details each implementation.

Evaluation metrics. We report accuracy as the mean and sample standard deviation over three inference runs. The mean weights all six benchmarks equally. Execution cost is the geometric mean over benchmarks of mean executor tokens relative to Direct; it excludes architect inference and training, whose cost we report separately. Architect and search settings are given in Appendix A.

Table 1: Accuracy (%, mean and standard deviation over three runs) with a shared Gemini 3.5 Flash executor. The first three benchmarks are short-horizon tasks that a single agent nearly solves; the last three are long-horizon and tool-intensive. Cost is the geometric mean of execution tokens relative to Direct. **Bold**: best; underline: second best. ‡Cost average excludes SpreadsheetBench: these methods answer in text, so they score zero there by construction and were not run (Appendix D).

	Short-horizon			Long-horizon, tool-intensive			**Aggregate**	
Method	GSM8K	HotpotQA	MBPP	Sheet	OfficeQA	GAIA	**Mean ↑**	**Cost ↓**
Prompting								
Direct	97.7 ±0.1	67.9 ±0.1	93.6 ±0.6	0.0	8.9 ±3.7	4.6 ±1.1	45.5	1.0
CoT	97.4 ±0.3	68.4 ±0.1	78.3 ±0.6	0.0	4.1 ±1.4	3.3 ±1.1	41.9	0.9‡
CoT (few-shot)	97.6 ±0.1	68.4 ±0.2	82.6 ±2.2	0.0	4.9 ±2.4	2.6 ±1.1	42.7	0.9
Self-Consistency	97.7 ±0.1	68.5 ±0.0	90.0 ±0.4	0.0	6.5 ±2.8	6.5 ±1.1	44.9	5.2‡
Plan-and-Solve	97.4 ±0.2	68.0 ±0.1	85.5 ±3.9	0.0	0.8 ±1.4	4.6 ±1.1	42.7	1.9‡
Self-Refine	97.8 ±0.2	68.5 ±0.0	86.3 ±1.6	0.0	3.3 ±2.8	2.6 ±1.1	43.1	2.3
Tree-of-Thoughts	97.8 ±0.1	68.3 ±0.0	91.6 ±0.2	0.0	0.8 ±1.4	5.2 ±2.3	43.9	17.4
Tool use and multi-agent								
ReAct	98.2 ±0.7	69.1 ±1.1	**97.1** ±0.6	84.4 ±5.5	16.3 ±3.7	33.3 ±7.1	66.4	12.3
LAP	95.5 ±0.4	17.0 ±0.3	7.3 ±0.8	6.1 ±4.3	1.6 ±1.4	6.5 ±3.0	22.3	7.8
AgentVerse	97.3 ±0.2	68.2 ±0.1	96.1 ±0.0	87.8 ±1.3	4.9 ±2.4	51.0 ±5.2	67.6	36.8
DSPy programs								
DSPy (unoptimized)	97.6 ±0.1	68.1 ±0.1	94.2 ±0.4	75.8 ±3.3	9.8 ±0.0	20.9 ±3.0	61.1	2.3
DSPy-MIPROv2	97.6 ±0.0	60.5 ±0.6	94.2 ±0.4	78.1 ±7.0	8.1 ±1.4	27.5 ±2.0	61.0	2.8
DSPy-GEPA	97.7 ±0.2	69.6 ±0.1	95.3 ±0.4	**95.3** ±1.7	11.4 ±2.8	29.4 ±2.0	66.5	2.5
Workflow optimization and query-adaptive								
AFlow	97.4 ±0.1	68.2 ±0.1	81.7 ±0.7	0.0	1.6 ±1.4	11.8 ±2.0	43.4	1.2
GPTSwarm	97.9 ±0.2	64.1 ±7.1	95.7 ±0.7	83.9 ±2.9	9.8 ±3.4	32.7 ±3.0	64.0	58.6
Trace	**98.4** ±0.2	**69.9** ±1.2	96.6 ±0.2	88.6 ±3.5	29.3 ±11.2	53.6 ±12.6	72.7	40.8
DyLAN	98.1 ±0.2	64.7 ±2.8	97.0 ±0.2	88.1 ±2.9	12.2 ±2.4	32.0 ±1.1	65.3	43.7
SHIFT-value	97.7 ±0.1	67.5 ±0.2	96.1 ±0.8	81.7 ±0.8	39.0 ±6.5	64.7 ±7.1	74.5	27.7
SHIFT-search	98.2 ±0.5	68.7 ±0.1	96.6 ±0.2	87.8 ±1.7	**59.3** ±5.1	**68.6** ±5.2	**79.9**	84.3

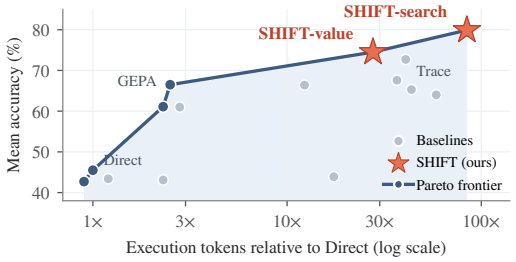

Figure 5: Accuracy–cost Pareto frontier for the six-benchmark methods in Table 1. Both SHIFT variants lie on the frontier.

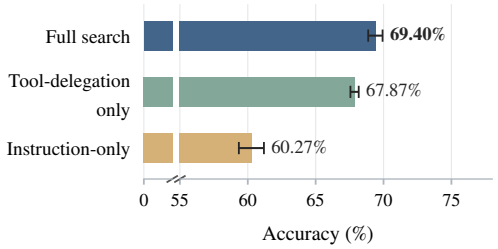

Figure 6: HotpotQA accuracy (500 questions; bars show SD over three runs). The broken axis omits 3–55%.

4.2 RQ1: Does Query-Specific Construction Improve Accuracy?

SHIFT constructs the most accurate harnesses. SHIFT-search attains the highest six-benchmark mean accuracy in Table 1, at 79.9%, an improvement of 7.2% over the strongest baseline, Trace (72.7%); a paired task-bootstrap 95% confidence interval runs from 4.42% to 9.88% (Appendix E.2). The gains concentrate on the two long-horizon, tool-intensive benchmarks, where no baseline is strong: every baseline stays below 30% on OfficeQA, yet SHIFT-search reaches 59.3% there and 68.6% on GAIA, exceeding Trace by 30.0% and 15.0%. SHIFT-value also outperforms every baseline on both, at 39.0% and 64.7%. These gains are broad rather than driven by a few tasks: against Trace, search wins on 24 OfficeQA questions and loses on four ($p = 2 \times 10^{-4}$, paired sign test), and wins on 19 GAIA questions and loses on seven ($p = 0.03$) (Appendix E.2).

Search pays off when two conditions hold: harness structure strongly affects success, and no existing design has already found the right one. OfficeQA and GAIA meet both: in training, adding agents or tools raises success by 50% and 34% (Figure 2), yet the best baseline reaches only 29.3% and 53.6%. Otherwise, SHIFT stays close to the leader. On GSM8K, HotpotQA, and MBPP, a

Table 2: Decoding comparison using the same trained architect within each benchmark. Cells report accuracy (%, mean ± sample standard deviation over three evaluation runs). Mean weights all six benchmarks equally; Cost is the geometric mean of execution-token ratios to Direct.

Mode	GSM8K	Hotpot	MBPP	SpreadSheet.	OfficeQA	GAIA	Mean	Cost
SHIFT-greedy	$98.0_{\pm 0.9}$	$69.2_{\pm 0.1}$	$96.4_{\pm 1.0}$	$87.5_{\pm 2.5}$	$52.8_{\pm 2.8}$	$75.2_{\pm 15.0}$	79.8	120.2
SHIFT-policy-veto	$98.1_{\pm 0.5}$	$69.2_{\pm 0.0}$	$96.4_{\pm 0.6}$	$85.6_{\pm 1.3}$	$59.3_{\pm 9.9}$	$66.0_{\pm 4.9}$	79.1	115.0
SHIFT-value-greedy	$98.0_{\pm 0.1}$	$64.6_{\pm 0.2}$	$95.6_{\pm 0.6}$	$58.6_{\pm 1.3}$	$10.6_{\pm 3.7}$	$34.0_{\pm 3.0}$	60.2	5.3
SHIFT-search	$98.2_{\pm 0.5}$	$68.7_{\pm 0.1}$	$96.6_{\pm 0.2}$	$87.8_{\pm 1.7}$	$59.3_{\pm 5.1}$	$68.6_{\pm 5.2}$	79.9	84.3
SHIFT-value	$97.7_{\pm 0.1}$	$67.5_{\pm 0.2}$	$96.1_{\pm 0.8}$	$81.7_{\pm 0.8}$	$39.0_{\pm 6.5}$	$64.7_{\pm 7.1}$	74.5	27.7

Table 3: Accuracy (%) of the harness chosen from identical candidate pools by the learned value V_θ (400 HotpotQA and 200 GSM8K questions, Gemma 3 27B executor).

Selection rule	HotpotQA	GSM8K
Lowest V_θ	$35.6_{\pm 1.2}$	$86.1_{\pm 0.2}$
Uniform	$39.1_{\pm 0.3}$	$89.7_{\pm 0.7}$
Second-highest V_θ	$37.7_{\pm 1.1}$	$89.9_{\pm 0.5}$
Highest V_θ (ours)	$\mathbf{49.6}_{\pm 0.4}$	$\mathbf{92.6}_{\pm 0.9}$

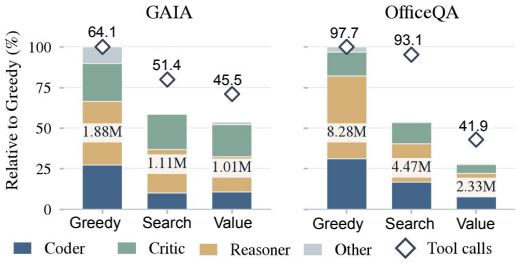

Figure 7: Tokens, tool calls, relative to Greedy. Labels mean tokens/task in millions inside bars and mean tool calls/task above diamonds.

single agent nearly matches the best method (16, 13, and 6 of the 17 baselines lie within 2%), so the executor rather than the harness sets the ceiling, and SHIFT-search is within 1.2% of it; without OfficeQA, SHIFT-search still has the highest mean (84.0% vs. 81.4% for Trace).

Ablation study on the action space. To test whether the gains require choosing agents, instructions, and tools together, we restrict SHIFT's action space at inference time on 500 HotpotQA questions with three runs each (Figure 6). Full search reaches 69.4%, compared with 67.9% when the architect may only delegate tools and 60.3% when it may only add instructions. Instructions tell an agent what to do, tools determine what it can do, and additional agents decide who does it; SHIFT gains from choosing these together rather than any one alone. The restricted variants start from a single Coder and cannot add roles, so the comparison measures the full construction space (Appendix E.3).

4.3 RQ2: How Does Harness Selection Balance Accuracy and Cost?

A cheaper operating point. SHIFT-value selects by predicted utility, which rewards cheaper harnesses, and sizes each harness to the query (Figure 8): it solves arithmetic and coding questions with one or two agents and few tool calls, and long-horizon tasks with larger harnesses that call tools extensively. It reaches 74.5%, above every baseline, while using 32% fewer execution tokens than Trace and one third those of SHIFT-search. Figure 5 places both variants on the accuracy–cost Pareto frontier: no baseline is both more accurate and cheaper than either. Against Trace, SHIFT-value is also more accurate on OfficeQA (+9.7%) and GAIA (+11.1%). One architect thus yields both the most accurate harness and a much cheaper one.

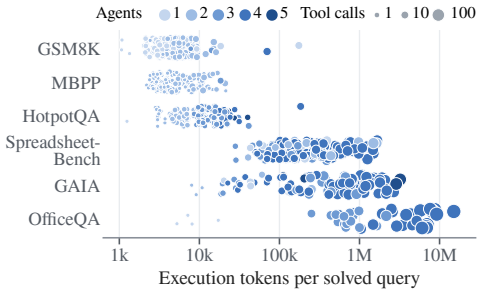

Figure 8: Solved test queries of SHIFT-value: execution tokens, agents in the harness (shade), and tool calls (size).

Search lowers execution cost without losing accuracy. To isolate search, we decode harnesses from the same trained architect in three simpler ways (Table 2): *greedy* construction applies the policy's most probable action at each step, *policy-veto* lets the value head reject low-utility policy proposals, and *value-greedy* moves to the one-action change the value head scores highest. Search matches greedy construction in mean accuracy (79.9% versus 79.8%) while using about 30% fewer execution tokens, and 27% fewer than policy-veto. Greedy decoding is itself a product of search: the policy is trained to reproduce search's visit counts (Section 3.4), so it inherits search's accuracy, while search at inference adds lookahead on predicted utility, which charges for execution cost. The search itself adds little: constructing a harness takes under a second, a small fraction of the time

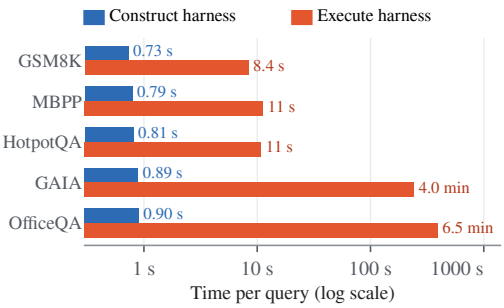

Figure 10: Time per query to construct a harness (one H100, 64 simulations) and to execute it (median over test queries, SHIFT-value).

needed to execute it (Figure 10). The savings are not simply fewer tool calls: on OfficeQA, tokens fall by 46% while tool calls barely change (Figure 7), and search is cheaper on 60–98% of questions. Value alone is not enough: value-greedy is the cheapest strategy but falls to 60.2%, since one-step lookahead cannot credit an agent whose benefit appears only after later actions (Appendix B).

Table 4: MATH-500 accuracy (%) by difficulty level, using the architect trained on GSM8K without retraining (Llama 4 Scout).

Configuration	Lv. 1–3	Lv. 4–5	All
Fixed single Coder	92.4	71.0	81.2
Uniform π, constant V	92.0	72.9	82.0
Learned π, constant V	91.6	74.0	82.4
Uniform π, learned V	91.6	71.8	81.2
SHIFT (learned π and V)	**92.4**	**75.6**	**83.6**

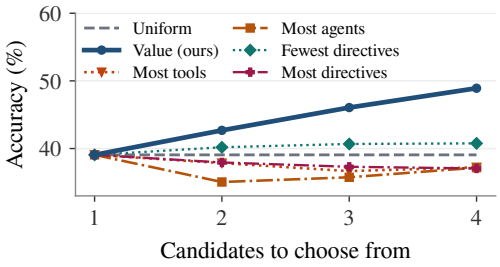

Figure 9: HotpotQA accuracy when choosing from k candidates.

4.4 RQ3: How Effectively Does Learned Value Guide Harness Selection?

We compare maximum-value and uniform selection over identical candidate pools, with values frozen before execution (Gemma 3 27B executor). Value selection raises accuracy by 10.5% on HotpotQA and 2.9% on GSM8K with half and one fifth of the execution tokens, respectively (Table 3; Appendix F.1). Its top choice beats the second-ranked candidate by 11.9% on HotpotQA, and the benefit grows as more candidates are considered (Figure 9). Fixed structural recipes fail: always picking the most agents, tools, or directives is worse than random selection, and picking the fewest directives barely beats it. Picking the fewest agents helps on HotpotQA but fails on long-horizon tasks (OfficeQA: 9.4% vs. 59.3% with four or more agents; Table 10).

Accurate and inexpensive choices. Because the value head is trained on a utility that rewards success and charges for cost, it learns to prefer the least expensive harness that still solves the query. This is what SHIFT-value exploits: it recovers several candidates from the search tree and executes the one the value head ranks highest, reaching the accuracy of the strongest baselines at a fraction of the cost of SHIFT-search (Section 4.3).

4.5 RQ4: Does the Architect Transfer to Harder Tasks?

An architect trained only on GSM8K transfers to the 500 MATH-500 problems (Llama 4 Meta (2025) executor), where SHIFT reaches 83.6% against 81.2% for a fixed single Coder. The gain concentrates on the hardest problems (Table 4): on levels 4–5, SHIFT reaches 75.6% against 71.0%, and both its learned policy and its learned value contribute (Appendix G).

5 Conclusion

We presented SHIFT, which casts harness design as search over executable graphs and learns, from executed harnesses, to predict which designs will succeed and at what cost. By moving execution out of the search loop, SHIFT constructs a harness for each query with a small local model in under a second on one GPU and executes only the selected harness. Across six benchmarks, it builds more accurate harnesses than 17 baselines on a shared executor, with the largest gains on long, tool-intensive tasks, and a value-based variant attains a higher mean accuracy than every baseline at lower cost than the strongest. These gains stem from choosing agents, instructions, and tools jointly, concentrate where structure matters and existing designs fall short, and carry over to harder problems without retraining. SHIFT shows how harness design can be learned from execution and adapted to individual queries. Training separate architects per benchmark incurs an upfront compute cost. Future work could develop an architect that generalizes across datasets and executors, reducing repeated training, and improve value estimation using outcomes averaged over executions.

AI USE STATEMENT

We used generative AI tools to polish the writing for readability and to check for typographical and grammatical errors in author-written text. All edits were reviewed by the authors, and no technical claim, result, or citation was altered. We take responsibility for the final content of this work.

ETHICS STATEMENT

Our work introduces an algorithmic contribution to query-specific agent harness design, aiming to improve task accuracy and computational efficiency. As such, we do not identify any explicit ethical concerns arising from this work.

REPRODUCIBILITY STATEMENT

Section 3 describes SHIFT's representation, search procedure, and learning objectives. Appendix A specifies the action vocabulary, tool availability, reward, and training and search settings, while Appendix B details harness selection and architect overhead. Dataset partitions, evaluation criteria, and baseline configurations are described in Appendices C and D. Appendices E, F, and G provide the additional evaluation protocols and uncertainty estimates supporting the main results. We plan to provide the full source code to reproduce the experiments following internal review.

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.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 17682–17690, 2024.

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.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, volume 2024, pp. 20094–20136, 2024.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and llms. *Advances in Neural Information Processing Systems*, 37:71596–71642, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Gemini Team. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023. URL https://arxiv.org/abs/2312.11805.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026. URL https://arxiv.org/abs/2607.02770.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 8154–8173, 2023.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Saiful Haq, Ashutosh Sharma, Thomas Joshi, Hanna Moazam, Heather Miller, et al. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations*, volume 2024, pp. 54928–54958, 2024.

Shiyuan Li, Yixin Liu, Qingsong Wen, Chengqi Zhang, and Shirui Pan. Assemble your crew: Automatic multi-agent communication topology design via autoregressive graph generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 23142–23150, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic llm-powered agent network for task-oriented agent collaboration. *arXiv preprint arXiv:2310.02170*, 2023.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems*, 37:94871–94908, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Meta. Llama-4-Scout-17B-16E-Instruct: Model card, 2025. URL `https://huggingface.co/meta-llama/Llama-4-Scout-17B-16E-Instruct`.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *International Conference on Learning Representations*, volume 2024, pp. 9025–9049, 2024.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366, 2024.

Krista Opsahl-Ong, Arnav Singhvi, Jasmine Collins, Ivan Zhou, Cindy Wang, Ashutosh Baheti, Owen Oertell, Jacob Portes, Sam Havens, Erich Elsen, et al. Officeqa pro: An enterprise benchmark for end-to-end grounded reasoning, 2026. *URL: https://arxiv. org/abs/2603.08655*, 2026.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. In *International Conference on Learning Representations*, volume 2025, pp. 3841–3865, 2025.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Aditya Taparia, Som Sagar, and Ransalu Senanayake. Learning to configure agentic AI systems. *arXiv preprint arXiv:2602.11574*, 2026. URL https://arxiv.org/abs/2602.11574.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 2609–2634, 2023.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Rui Ye, Shuo Tang, Rui Ge, Yaxin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. Mas-gpt: Training llms to build llm-based multi-agent systems. *arXiv preprint arXiv:2503.03686*, 2025.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15549–15572, 2025.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025a.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025b.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*, 2024.

APPENDIX

The appendix is organized in three parts. Appendix A specifies SHIFT's action space, architect, search, reward, training, and training cost, and Appendix B compares the ways of decoding a harness from the trained architect, shows how cost and harness size vary across queries, and measures architect overhead. Appendices C and D describe the benchmarks, evaluation protocol, and baselines. Appendices E, F, and G give further results for Sections 4.2, 4.4, and 4.5.

A METHOD AND IMPLEMENTATION DETAILS

This appendix specifies the components that Sections 3.1–3.4 describe: the action space over harnesses, the architect's observation, the batched form of the tree search, the execution reward that defines the architect's learning target, and the training procedure.

A.1 ACTION SPACE

Every search begins from the same minimal harness s_0, a single Coder agent without tools, so that any additional structure must be justified by search rather than assumed. From s_0, SHIFT grows a harness through a vocabulary of 40 actions (Table 6) that mirror the three components of a harness in Section 3.1: ten structural actions that add an agent or a feedback loop, grant the full tool set, or stop; fifteen directives that append an instruction to an existing agent; and fifteen tool grants that give an existing agent a single tool. The feasible set $\mathcal{A}(s)$ contains only actions that change the harness and keep it executable: directives and tool grants require their target agent to exist, and a tool can be granted only if the benchmark provides it. Because every action changes a single component, a path through the search tree records exactly which decisions produced the final harness. The library is shared across benchmarks: every structural action and directive is available on every benchmark, and only tool grants depend on the tools a benchmark provides, so the architect must learn from execution which components help where. The resulting design space is large: counting every combination of at most ten actions that respects these dependencies gives about 1.5×10^7 distinct harnesses, somewhat fewer on benchmarks that offer fewer tools. Table 5 summarizes the search configuration used in all experiments.

Table 5: Search settings shared by all benchmarks.

Setting	Value
Actions in the vocabulary	40
Branches kept per node, K_{exp}	10
Maximum number of actions, H	10
Search simulations, B	64
Exploration constant, c_{puct}	2.5
Policy temperature, T_π	1
Evaluated per architect forward pass	8
Architect input length	2,048 tokens
SHIFT-value: candidates ranked	5
SHIFT-value: re-search below value	0.55
Policy-veto: value threshold	0.55

Tool availability. GSM8K, HotpotQA, OfficeQA, and SpreadsheetBench provide a file reader, file writer, Python execution, and a calculator. MBPP provides the first three, and GAIA provides all four together with web search and a web scraper. When an agent is added, it receives a small default tool set that fits its role, restricted to the tools the benchmark provides: Researchers and Critics get the file reader, Reasoners the calculator, and Verifiers Python execution and the calculator.

A.2 ARCHITECT, SEARCH, AND REWARD

Observation. The architect conditions on the information that determines which action is useful: a short description of the benchmark, its available tools, the query, the feasible actions $\mathcal{A}(s)$, and a text serialization of the current harness graph that lists each agent's role, tools, and outgoing edges. Because the feasible actions are part of the input, the policy head scores exactly the actions that can be taken. In-

Table 7: Reference for the token and latency penalties, matched to cost of each benchmark.

Benchmark	Tokens ρ_t	Latency ρ_ℓ (s)
GSM8K	30,000	25
HotpotQA	40,000	30
MBPP	60,000	60
SpreadsheetBench	250,000	100
OfficeQA	1,500,000	400
GAIA	300,000	200

Table 6: The action vocabulary. Directives are appended at most once to their target agent.

ID	Target	Action
1–6	Graph	Add a Planner, Researcher, Reasoner, Critic, Verifier, or Aggregator agent.
7–8	Graph	Add a feedback loop from the Verifier or the Critic back to the executing agent.
9	Graph	Give every executing agent the full tool set.
10	Graph	Stop and return the current harness.
11	Coder	Think step-by-step before generating code.
12	Coder	Write clean, concise code without extra comments.
13	Coder	Ensure all spreadsheet formula generation conforms strictly to Excel standards.
14	Coder	Include error handling for division by zero and empty datasets.
15	Reasoner	Think step-by-step and show your reasoning path.
16	Reasoner	Provide a direct, concise analytical answer.
17	Critic	Explicitly check units (e.g., millions vs. billions) and bounds.
18	Verifier	Verify outputs using code execution whenever possible.
19	Planner	Decompose the query into detailed, granular sub-tasks.
20	Researcher	Align table row and column headers carefully before aggregating values.
21	Aggregator	Provide a high-level summary followed by the final answer.
22	Aggregator	Output the final answer only, without any prefix.
23	Reasoner	Write out symbolic formulas and substitute variables step-by-step.
24	Critic	Check that the answer directly addresses the query in the required format.
25	Verifier	Write assertions that check the output's type, bounds, and format.
26–30	Coder	Grant web search, web scraper, file reader, file writer, or calculator.
31–34	Reasoner	Grant web search, web scraper, file reader, or calculator.
35–36	Researcher	Grant web search or web scraper.
37–38	Critic	Grant file reader or file writer.
39–40	Verifier	Grant Python execution or calculator.

puts are limited to 2,048 tokens, and long queries retain their beginning and end. Both heads read the representation of the final token, $h_\theta(q, s)$.

Batched tree search. Evaluating one leaf per architect call would leave the accelerator largely idle, so SHIFT evaluates up to eight leaves in a single forward pass. Without further care, parallel descents would all follow the same promising branch. We prevent this with a virtual loss: while a leaf awaits evaluation, every node on its path counts it as a visit of value zero, which temporarily lowers the branch's score and steers other descents elsewhere. Writing $L(s)$ and $L(s, a)$ for these pending visits and $W(s, a)$ for the sum of values backed up through action a, the statistics in Equation (3) are $\widetilde{N}(s) = N(s) + L(s)$, $\widetilde{N}(s, a) = N(s, a) + L(s, a)$, and

$$\widetilde{Q}(s,a) = \begin{cases} W(s,a)/\widetilde{N}(s,a), & \widetilde{N}(s,a) > 0, \\ \mathrm{clip}\left(Q(s) - 0.1,\, 0,\, 1\right), & \widetilde{N}(s,a) = 0,\ N(s) > 0, \\ 0.5, & \text{otherwise}, \end{cases} \tag{6}$$

where $Q(s)$ is the mean value backed up through s. Untried actions are thus initialized slightly below their parent's value, a conservative choice that keeps search on actions the policy favors until alternatives are evaluated. Virtual visits are removed as soon as the predicted values are backed up.

Execution reward. The value head learns to predict a utility that rewards task success and penalizes the resources a harness consumes, so that search favors the least expensive harness that solves the query. For an executed harness with task score $u \in [0, 1]$ (binary success when the benchmark provides no graded score), the reward and its rescaled utility are

$$R = \alpha u - b - \sum_x \gamma_x r_x, \qquad z = \frac{\mathrm{clip}(R, -1, 1) + 1}{2}, \tag{7}$$

where $z \in [0, 1]$ is the value head's target and the sum runs over five penalties, each in $[0, 1]$. With t and ℓ the execution tokens and latency, and ρ_t and ρ_ℓ benchmark-specific reference scales (Table 7),

$$r_{\mathrm{tok}} = \frac{t}{t + \rho_t}, \quad r_{\mathrm{lat}} = \frac{\ell}{\ell + \rho_\ell}, \quad r_{\mathrm{tool}} = \min\left(1, \tfrac{n_{\mathrm{tool}}}{6}\right), \quad r_{\mathrm{agent}} = \min\left(1, \tfrac{|\mathcal{V}|-1}{7}\right), \tag{8}$$

Table 8: Training hyperparameters for the Gemma-4-E2B architect.

Setting	Value
LoRA rank / scale / dropout	16 / 32 / 0.05
LoRA target layers	attention query, key, value, output
Optimizer	AdamW, learning rate 10^{-4}, weight decay 0.01
Learning-rate schedule	cosine, minimum 10^{-5}
Gradient clipping	1.0
Epochs	4
Entropy weight β / value weight λ_v / ranking weight λ_r	0.01 / 1 / 1
Ranking margin m / minimum utility gap δ	0.05 / 0.05
Root exploration noise	Dirichlet(0.35), mixing weight 0.25
Replay buffer	256 queries, prioritized ($a = 0.6$, $b = 0.4$)

and $r_{\text{timeout}} \in \{0, 1\}$ indicates a timeout. The token and latency penalties saturate smoothly relative to their reference scales, which places benchmarks of very different cost, from GSM8K to OfficeQA, on a common scale while retaining sensitivity to cost beyond the reference. The tool penalty counts the distinct tool types granted, n_{tool}, rather than tool calls, so it charges for the capabilities a harness is given; tools essential to a benchmark, such as the file reader for OfficeQA, are exempt. The agent penalty grows with each agent beyond the first, $|\mathcal{V}|$ being the number of agents. We use $\alpha = 1.5$, $b = 0.5$, and weights γ of 0.15 (tokens), 0.10 (latency), 0.10 (tools), 0.08 (agents), and 0.15 (timeout). Since the penalties sum to at most 0.58, every successful harness receives a utility of at least 0.71 and every failed harness at most 0.25: success always dominates, and among harnesses that succeed, the cheaper one is preferred.

Algorithm 2 Training the SHIFT architect

Require: training queries $\mathcal{Q}_d$, architect θ, executor, seed harness s_0, budgets B, D, K_{exec}
$\mathcal{Z} \leftarrow \varnothing$ ▷ executed harnesses and utilities
for each epoch **do**
  for $q \in \mathcal{Q}_d$ in random order **do**
    $\mathcal{T} \leftarrow$ search from s_0 with budget B and root noise (Algorithm 1)
    sample a path by visit counts; record policy targets along it
    execute up to K_{exec} distinct harnesses from $\mathcal{T}$; compute utilities z (Eq. 7)
    add the executed harnesses to $\mathcal{Z}$ and update θ with $\mathcal{L}_{\text{online}}$
  end for
  for each replay step **do**
    sample executed harnesses and same-query pairs from $\mathcal{Z}$
    update θ with the value loss plus $\lambda_r \mathcal{L}_{\text{rank}}$
  end for
end for
return θ

A.3 Training

Training alternates between search and learning (Algorithm 2). For each training query, the architect searches from s_0 with Dirichlet noise added to the root priors to encourage exploration, samples a construction path in proportion to visit counts, and executes the harness at its end together with up to K_{exec} other distinct harnesses recovered from the most-visited branches. As in AlphaZero Silver et al. (2017), the visit distributions along the path supervise the policy, while each executed harness supplies a measured utility for the value head. Executing several harnesses per query is what enables the ranking objective, since it yields pairs of harnesses for the same query whose utilities can be compared. Examples enter a prioritized replay buffer that favors states whose values are currently mispredicted, and the architect is updated online with $\mathcal{L}_{\text{online}}$ (Equation 4). At the end of each epoch, a replay phase revisits all executed harnesses and optimizes the value loss jointly with $\mathcal{L}_{\text{rank}}$ (Equation 5) on same-query pairs. Table 8 lists the hyperparameters. Training also reshapes the harnesses the architect builds (Figure 11): by the final epoch, Planners appear in most

	Planner	Researcher	Reasoner	Critic
GSM8K	90 +48	68 +22	5 −33	66 −1
MBPP	89 +29	1 −5	62 +3	68 +8
HotpotQA	90 +22	65 −22	36 +10	82 +22
SpreadsheetBench	92 +24	2 −10	72 +21	88 +10
GAIA	86 +9	17 −6	50 +40	79 +16
OfficeQA	79 +7	0 −15	40 +21	73 +15

Figure 11: How harness composition changes over training. Each cell gives the share (%) of final-epoch training harnesses that contain a role, with the change since the first epoch below.

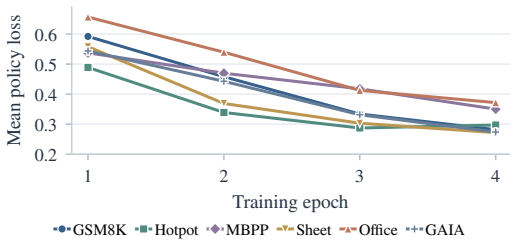

Figure 12: Mean policy loss over four training epochs, one run per benchmark, averaging records with nonzero total loss. Hotpot, Sheet, and Office denote HotpotQA, SpreadsheetBench, and OfficeQA.

harnesses on every benchmark, while Researchers are largely dropped on MBPP, SpreadsheetBench, and OfficeQA and Reasoners become common on GAIA.

Training cost. Each architect trains for four epochs, and each epoch searches every training question once and executes up to five distinct harnesses from its tree. This amounts to 1,092 executions on OfficeQA, 1,611 on GAIA, 2,400 on MBPP, 4,250 on SpreadsheetBench, and about 9,400 each on GSM8K and HotpotQA. In executor tokens, training one architect costs about as much as answering 440 (OfficeQA) to 6,200 (HotpotQA) test questions with SHIFT-search. This cost is paid once per benchmark and executor and is then amortized over every later query, which each need only one execution.

Training curves. Figure 12 tracks the policy objective over training. Each architect sees every training question once per epoch, with the same questions in all four epochs (Table 9), and we report the mean policy loss over the updates of each epoch. The loss falls by 35% to 52% on every benchmark. Because the policy targets are produced by search with the current architect, they sharpen as the architect improves; the decreasing loss therefore shows the policy increasingly anticipating the outcome of its own search, which is what allows search to concentrate its budget on promising actions.

B Harness Selection Strategies

The architect provides two learned signals, a policy over actions and a value over harnesses, and a harness can be decoded from them in several ways, much as a language model can be decoded greedily or with search. We compare five decoding strategies that share the same trained architect for each benchmark, start from s_0, and apply at most ten actions. They differ only in which signal chooses each action and whether actions are chosen with lookahead.

- **Greedy** applies the policy's most probable action at each step until the policy chooses to stop. It isolates the policy.
- **Policy-veto** lets the policy propose actions in order of probability and the value head accept the first of the top four whose predicted utility is at least 0.55, falling back to the policy's first choice. The value can thus overrule the policy, but only one step at a time.
- **Value-greedy** scores every one-action change with the value head and moves to the best one while it improves the current harness. It isolates the value without lookahead.
- **SHIFT-search** runs Monte Carlo tree search with 64 simulations and executes the harness at the end of the most-visited path, combining both signals with lookahead.
- **SHIFT-value** runs the same search, recovers up to five distinct harnesses from the most-visited branches, and executes the one with the highest predicted utility. If that utility is below 0.55, it searches once more with 192 additional simulations and chooses among both sets of candidates.

Comparing these strategies separates the contributions of the policy, the value, and lookahead (Table 2). Search matches the accuracy of greedy construction while using 30% fewer execution tokens,

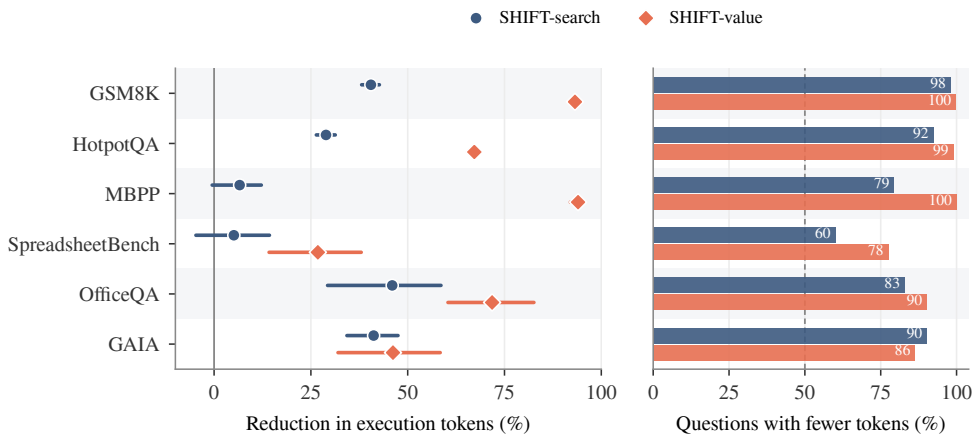

Figure 13: Execution-token savings relative to greedy construction, paired by question. Left: reduction in mean execution tokens, with 95% bootstrap intervals over questions. Right: share of questions on which each strategy uses fewer tokens than greedy construction

because it weighs each action by the utility of the harnesses it leads to. Value-greedy is the cheapest strategy but loses about 20% in mean accuracy: an agent or tool whose benefit appears only after later actions looks unattractive one step ahead. The value head is therefore most effective when combined with lookahead, as in SHIFT-search, or used to rank complete harnesses, as in SHIFT-value. Figure 7 shows where search saves tokens relative to greedy construction.

B.1 Cost savings across questions

Aggregate savings could in principle come from a few very expensive questions. To test whether they are broad, we pair every evaluation question across strategies and compare the mean execution tokens of its three runs (Figure 13). Search uses fewer tokens than greedy construction on 60% to 98% of questions, depending on the benchmark, and SHIFT-value does so on 77.5% to 100%. The savings are largest on the most expensive benchmarks, where search cuts mean tokens by 46% on OfficeQA and 41% on GAIA.

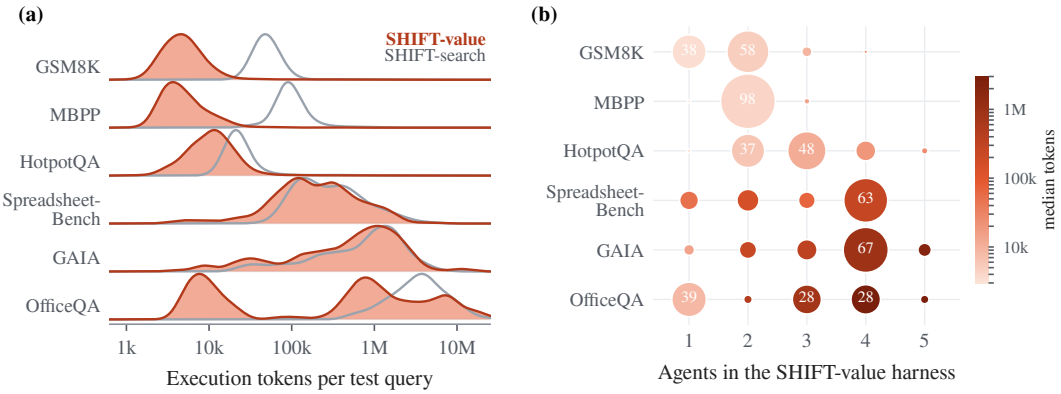

Figure 14: Cost and harness size per test query (all test runs). (a) Distribution of execution tokens per query for SHIFT-search and SHIFT-value. (b) Harnesses chosen by SHIFT-value: bubble area is the share of test queries whose harness has that many agents (labels for shares of at least 25%), and color is the median execution tokens of those queries.

Cost and harness size per query. Figure 14 shows how the two operating points spend tokens query by query. SHIFT-search uses a similar amount of structure on every query of a benchmark, so its cost is concentrated around one value. SHIFT-value chooses among harnesses of different sizes for

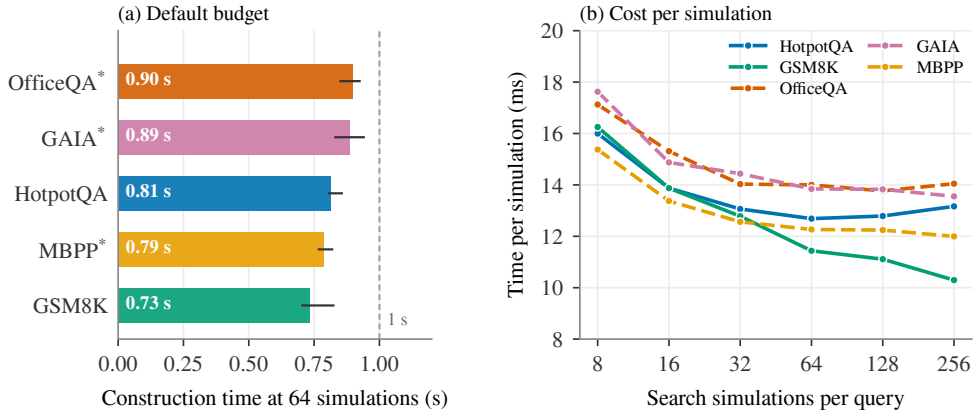

Figure 15: Architect construction cost on one H100 GPU (median over ten questions, three repeats each). (a) Time to construct one harness at the default budget of 64 simulations; whiskers show the interquartile range. (b) Time per simulation as the budget grows. Benchmarks marked * (dashed in b) use the HotpotQA architect with the benchmark's tools.

each query: on GSM8K and MBPP it answers almost every query with one or two agents at about a tenth of SHIFT-search's cost, on HotpotQA it mostly uses two or three, and on SpreadsheetBench and GAIA it builds harnesses with four or more agents, as search does. On OfficeQA its choices split between a single agent and three or four agents, which gives its cost distribution two modes.

B.2 Architect overhead

Because SHIFT replaces executor calls with local inference, its design cost per query is the time the architect spends searching. We measure the complete construction call on a single H100 NVL GPU for ten questions on each of five benchmarks, with three repeats each and model loading excluded (Figure 15). At the default budget of 64 simulations, constructing a harness takes a median of 0.81 seconds on HotpotQA and 0.73 seconds on GSM8K, and between 0.79 and 0.90 seconds on OfficeQA, GAIA, and MBPP. Greedy construction takes 0.26 seconds, and SHIFT-value, including its occasional second search, 0.86 and 0.78 seconds on HotpotQA and GSM8K. Larger budgets remain affordable: each simulation costs 10 to 14 milliseconds at budgets of 64 and above, slightly less than at small budgets as fixed per-call overhead is amortized, so construction time grows linearly with the budget and reaches 2.6 to 3.6 seconds at 256 simulations. Because simulations are evaluated in batches by a 2B-parameter model, the design cost stays below one second per query at the default budget, whereas executing a single multi-agent harness on OfficeQA or GAIA consumes roughly one to eight million executor tokens (Figure 7).

C Benchmarks and Evaluation Protocol

Benchmark selection. The six benchmarks are chosen to span the range over which the value of a harness should vary. At one end, GSM8K, HotpotQA, and MBPP pose short, self-contained problems that a single well-prompted agent largely solves. At the other, SpreadsheetBench, OfficeQA, and GAIA require reading and producing files, querying large document collections, or browsing the web over many steps, so success depends on which agents are present and which tools they can use. Evaluating on both ends tests whether SHIFT adds structure where it helps and withholds it where it does not. Table 9 summarizes each benchmark, its success criterion, and its data split.

GSM8K uses its official test set (1,319 questions) and HotpotQA its official distractor development set (7,405), since HotpotQA test labels are not public; from each official training set we draw 600 questions, 472 for training and 128 for validation. MBPP keeps its official splits. SpreadsheetBench (400 tasks), OfficeQA Pro (133 questions), and GAIA (the 165 tasks of its public validation release) provide no training split, so we divide each at random into training, validation, and test sets in proportions of 55%, 15%, and 30%. Test questions are never used for training or model selection, and all methods are evaluated on the same test sets.

Table 9: Benchmarks, success criteria, and data splits (number of tasks). Training, validation, and test sets are disjoint.

Benchmark	Task	Success criterion	Train	Val	Test
GSM8K	Grade-school math word problems	Final number equals the reference	472	128	1,319
HotpotQA	Multi-hop questions over supplied passages (distractor setting)	Exact match after standard answer normalization	472	128	7,405
MBPP	Short Python functions from a description (sanitized set)	Generated code passes all unit tests	120	43	257
SpreadsheetBench	Spreadsheet manipulation from natural-language instructions	Output workbook matches the reference on every test case	220	60	120
OfficeQA Pro	Questions over a corpus of 697 U.S. Treasury Bulletins	Answer matches the reference within 1%	73	19	41
GAIA	General assistant tasks needing web search, files, and reasoning	Normalized answer matches the reference	90	24	51

Metrics. Accuracy is each benchmark's success criterion averaged over the three test runs. Every test task counts: failures and timeouts are scored as incorrect and are never removed. The mean in Table 1 weights benchmarks equally, so that HotpotQA's 7,405 questions do not outweigh OfficeQA's 41. Execution cost counts the tokens consumed by the executor across all agents of the executed harness, relative to Direct. Because this ratio spans about two orders of magnitude across methods, we aggregate it with a geometric mean, which weights a halving of cost equally on every benchmark. Architect computation is reported separately (Appendix B.2).

D BASELINES

The baselines fall into the four groups of Table 1. Prompting methods fix a single-agent harness. Tool-using and multi-agent frameworks (ReAct, LAP, and AgentVerse) fix a harness with tools or several agents; LAP (LLM as a Policy) runs a tree search over tool-using actions in which the executor LLM proposes and scores the actions. The DSPy programs run unoptimized or with instructions and demonstrations tuned by MIPROv2 or GEPA. Workflow optimizers (AFlow, GPTSwarm, and Trace) search for one structure per benchmark, and DyLAN composes its agent team for each query. SHIFT differs from all of them in constructing the structure, instructions, and tool access of the harness jointly for each query.

To isolate the effect of harness design, every baseline runs in the same execution and grading framework as SHIFT, with the same Gemini 3.5 Flash executor, the same test sets, and the same answer checking. Differences in Table 1 therefore reflect how each method organizes the executor rather than the model or the grader. Every cell averages three runs, except GPTSwarm on HotpotQA and OfficeQA, which average two.

Zero scores on SpreadsheetBench. A SpreadsheetBench task counts as solved only if code writes a modified workbook to disk and that workbook matches the reference on every test case. The task therefore cannot be completed without file and code-execution tools. Methods that return their answer as text (Direct, CoT (few-shot), Self-Refine, Tree-of-Thoughts, and our AFlow configuration) never produce the output workbook and score zero. CoT, Self-Consistency, and Plan-and-Solve also answer in text, so we assign them zero without running them; the zero enters their Mean, and their Cost excludes SpreadsheetBench (‡ in Table 1). Methods that can equip agents with file and code tools, including ReAct, the DSPy programs, and SHIFT, can solve these tasks.

E ADDITIONAL RESULTS ON HARNESS ACCURACY

This appendix supports the accuracy results of Section 4.2. We first show that the benefit of structure differs across benchmarks, which is the premise of per-query construction; we then test the gains over the strongest baseline question by question, and finally examine which parts of the action space the gains require.

E.1 Structure sensitivity

Per-query construction is only worthwhile if the right amount of structure differs from task to task. We test this premise on the training runs. During training, search executes many different harnesses on the same questions, which yields a large and varied sample of harnesses run by the Gemini 3.5 Flash executor. For each benchmark, we group these executions by their amount of structure along one action type and compare success between the two extremes: one agent versus four or more, no tools versus every tool the benchmark offers, and no appended directives versus two or more. Figure 2 reports the resulting change in utility, and Table 10 gives the success rates and group sizes. All nine contrasts on SpreadsheetBench, GAIA, and OfficeQA are significant (two-sided Fisher exact test, $p \leq 0.016$), and none on GSM8K. The effect of structure ranges from none on GSM8K to 50% on OfficeQA, and it differs by action type within a benchmark: on SpreadsheetBench, tools matter most, and on OfficeQA, agents. Because search chose which harnesses to run, this comparison is observational: it shows where structure and success go together rather than the effect of adding structure to a fixed harness.

Table 10: Success rates (%) of executions with the least and most of each structure type; the number of executions is in parentheses.

	Agents		Tools		Directives	
Benchmark	1	≥ 4	none	all	0	≥ 2
GSM8K	98.2 (114)	98.4 (835)	98.4 (127)	97.8 (965)	98.0 (665)	97.9 (419)
MBPP	94.1 (34)	99.0 (210)	95.6 (68)	99.4 (319)	98.0 (247)	100.0 (37)
HotpotQA	75.6 (90)	84.3 (1,116)	72.4 (76)	83.7 (1,611)	79.6 (554)	83.1 (278)
SpreadsheetBench	55.3 (38)	82.8 (476)	44.8 (29)	87.3 (700)	65.4 (208)	83.0 (406)
GAIA	40.0 (25)	73.9 (134)	35.3 (17)	66.9 (290)	50.0 (90)	72.2 (158)
OfficeQA	9.4 (32)	59.3 (59)	16.0 (25)	46.8 (190)	26.2 (61)	54.6 (97)

E.2 Paired comparison with Trace

A higher mean accuracy can arise from a few benchmarks or a few questions, so we compare SHIFT-search with Trace, the strongest baseline, question by question. For each benchmark, we average the three runs of each question and resample questions with replacement (20,000 bootstrap draws), weighting benchmarks equally in the six-benchmark mean. SHIFT-search improves the mean by 7.2% (95% interval [4.4, 9.9]), well above zero. The improvement comes from OfficeQA (+30.1%, [17.1, 43.1]) and GAIA (+15.0%, [5.9, 24.8]), while the two methods are within about one percent on the other four benchmarks.

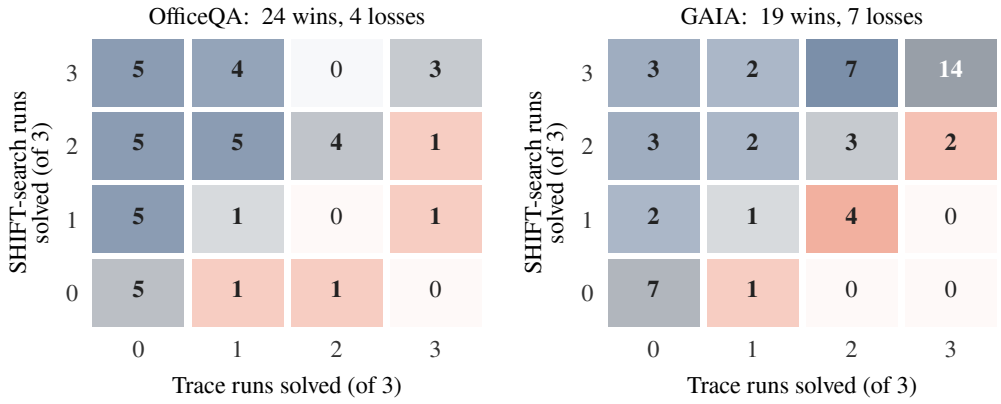

Figure 16: Question-level comparison with Trace, the strongest baseline. (a) Accuracy difference of SHIFT-search, with paired 95% bootstrap intervals over questions; green marks intervals above zero. (b) Questions on OfficeQA (41) and GAIA (51) that each SHIFT variant solves in more (win), as many (tie), or fewer (loss) of three runs than Trace.

The gains are also broad within these benchmarks (Figure 16). Counting successes over three runs, SHIFT-search solves more often than Trace on 24 OfficeQA questions and less often on only 4, and on GAIA the counts are 19 and 7; SHIFT-value likewise wins more questions than it loses on both benchmarks (15 against 9 on OfficeQA and 18 against 7 on GAIA). SHIFT-search also solves five OfficeQA questions and three GAIA questions in every run on which Trace fails in every run, with no case in the opposite direction.

Robustness of the ranking. Two alternative aggregations leave the ranking unchanged. Excluding SpreadsheetBench, which three baselines did not run, SHIFT-search reaches 78.3% and SHIFT-value 73.0%, against 69.6% for Trace, the strongest baseline. Removing OfficeQA, the benchmark with the largest gain, SHIFT-search still leads with 84.0%.

E.3 Action-space restrictions

To test whether the gains require choosing structure, instructions, and tools together, we restrict the actions available to the trained HotpotQA architect at inference time, without retraining, on 500 HotpotQA test questions with three runs each and 64 simulations. *Directives only* allows only instruction actions (actions 10–25 in Table 6), and *tool grants only* allows only tool actions (actions 10 and 26–40); both keep the single Coder, since they cannot add agents. Each restriction lowers accuracy (Table 11). Paired over questions with 20,000 bootstrap draws, directives alone lose 9.1% (95% interval [6.4, 12.0]) and tool grants alone lose 1.5% ([0.1, 3.1]); both intervals exclude zero. Tools recover most of the accuracy on this retrieval-heavy benchmark, but only the full action space, which can also add agents and instructions, reaches the best result.

Table 11: HotpotQA accuracy under restricted action spaces (500 questions, three runs each).

Actions allowed	Accuracy (%)	Change vs. full (%)
All actions	69.40	–
Tool grants only	67.87	−1.53
Directives only	60.27	−9.13

F Learned Value for Harness Selection

This appendix supports Section 4.4. SHIFT-value relies on the value head to choose among candidate harnesses without executing them, so we evaluate the value head directly as a ranker, separately from the search that proposes candidates.

F.1 Candidate ranking

Setup. Evaluating a ranker requires knowing how every candidate would have performed. We therefore build a pool of distinct candidate harnesses for each of 400 HotpotQA and 200 GSM8K questions from several construction strategies (greedy construction, search with 8 and 64 simulations, search with a constant value, and a fixed reference harness), execute every candidate twice with a Gemma 3 27B executor, and score each candidate with the value head before any outcome is inspected. Each pool holds four or five candidates. For question i with pool $\mathcal{C}_i$, let v_{ic} be the predicted utility of candidate c and y_{ic} its mean success. Value selection and uniform selection achieve

$$A_i^{\text{value}} = \frac{1}{|M_i|} \sum_{c \in M_i} y_{ic}, \quad M_i = \arg\max_{c \in \mathcal{C}_i} v_{ic}, \qquad A_i^{\text{uniform}} = \frac{1}{|\mathcal{C}_i|} \sum_{c \in \mathcal{C}_i} y_{ic}, \tag{9}$$

where ties in predicted utility are averaged. Uniform selection is computed in expectation, so it adds no sampling noise, and the two selectors are compared on identical pools and outcomes.

The value head selects more accurate and cheaper harnesses. Value selection outperforms uniform selection by 10.5% on HotpotQA (95% bootstrap interval over questions [7.5, 13.6]) and by 2.9% on GSM8K ([0.8, 5.0]). It does so while choosing much cheaper harnesses (Figure 17): the selected harness uses 9.5k execution tokens on HotpotQA against 19.5k for a uniform choice, and 1.8k against 8.8k on GSM8K. Because the value head is trained on a utility that rewards success and charges for cost (Equation 7), it learns to prefer the least expensive harness that still solves the query, which is the behavior that gives SHIFT-value its low execution cost.

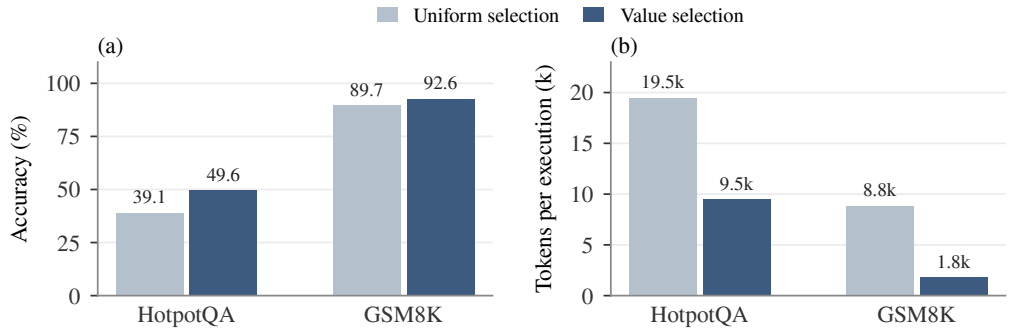

Figure 17: Choosing among the same candidate harnesses with the value head versus uniformly at random (400 HotpotQA and 200 GSM8K questions, Gemma executor). (a) Accuracy. (b) Execution tokens of the selected harness.

More candidates, better choices. For Figure 9, we average each selector over every subset of k candidates from each pool, for $k = 1, \ldots, 4$, so that every question contributes at every k. Uniform selection has the same expected accuracy for every k, so a rising value curve reflects better choices rather than easier questions. Value selection improves steadily as more candidates become available, from 39.1% with one candidate to 42.7%, 46.1%, and 48.9% with four on HotpotQA, and from 89.7% to 92.4% on GSM8K.

G Transfer to Harder Tasks

This appendix supports Section 4.5. A useful architect should carry over to tasks harder than those it was trained on; we test this by applying the GSM8K architect, without retraining, to MATH-500.

We apply the GSM8K architect, without further training and without access to MATH answers, to all 500 problems of MATH-500 Hendrycks et al. (2021); Lightman et al. (2024), which span seven subjects and five difficulty levels. No MATH-500 question appears in the GSM8K splits. Each problem is executed once with Llama 4 Scout at temperature 0.2, and answers are graded with the Math-Verify library.

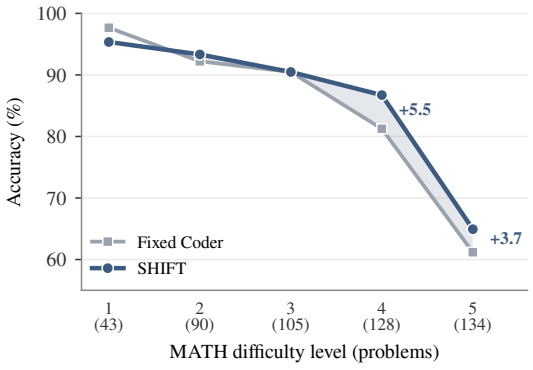

Figure 18: MATH-500 accuracy by difficulty level for SHIFT, using the GSM8K architect without retraining.

SHIFT reaches 83.6%, against 81.2% for a fixed single Coder, and the gain is concentrated where it should be (Figure 18). On difficulty levels 1–3, which a single agent already solves about 92% of the time, the two are tied. On the two hardest levels, SHIFT improves accuracy by 5.5% and 3.7%, reaching 75.6% against 71.0% over these 262 problems; paired over problems, this difference of 4.6% has a 95% bootstrap interval of [0.4, 9.2]. The harnesses built by an architect trained only on grade-school arithmetic thus help most on the hardest problems, where a single agent falls short.

Table 12 further shows that the complete architect, with both its learned policy and its learned value, is the most accurate configuration: replacing either with an uninformed version lowers accuracy.

Table 12: MATH-500 accuracy with Llama 4 Scout, using the GSM8K architect without retraining (500 problems per row, one execution each). Uniform policy: feasible actions receive equal prior probability. Constant value: every harness receives a predicted utility of 0.5.

Configuration	Accuracy (%)	Executor tokens / problem
SHIFT (learned policy, learned value)	**83.6**	8,049
Learned policy, constant value	82.4	7,614
Uniform policy, constant value	82.0	6,283
Uniform policy, learned value	81.2	5,585
Fixed single Coder	81.2	1,064