RELEVOLVE: LEARNING WHAT TO EVOLVE, EVOLVING WHAT TO LEARN

Anonymous authors
Paper under double-blind review

ABSTRACT

Automated harness optimization improves LLM agents by iteratively revising the programs that govern prompting, memory, tool use, and control flow. Existing harness optimizers use prior harnesses and evaluation feedback to decide how to modify the next candidate, but the unresolved improvement directions that should guide subsequent evolution typically remain implicit. We introduce *RelEvolve*, which represents these directions as inquiries and couples inquiry updating with harness evolution: inquiries guide which candidate to continue from and what change to make, while the resulting evaluation provides evidence to refine the inquiry state for later evolution rounds. RelEvolve realizes this coupling through a *Harness Lineage* that records executable inheritance, an *Inquiry Graph* that maintains inquiry state across branches, and a *Relational Trace* that links the inquiries guiding each evolution to the resulting harness change and evaluation evidence. Across five benchmarks and two LLM backbones, RelEvolve yields a 4.4% absolute improvement in held-out test performance over the strongest evaluated baseline in each setting and a 12.5% absolute improvement over the unoptimized harness, averaged across settings. On HotpotQA, it reaches 89.84 and 91.24 F1 with GPT-5.6 Luna and GLM-5.3-Flash, respectively. Three independent search seeds on HotpotQA and InterCode-SQL maintain this improvement in these two representative settings, and on HotpotQA, ablations further indicate that weakening the coupling between inquiry updating and harness evolution lowers performance.

1 INTRODUCTION

Large language models depend not only on model weights but also on the executable harness that surrounds them. A harness determines how the model is prompted, what information it retrieves and retains, which tools it invokes, and how these operations are composed into an executable procedure. Recent work has moved beyond optimizing isolated prompts toward automatically designing and improving complete agent programs and workflows (Hu et al., 2025; Zhang et al., 2025b; Zhao et al., 2026; Shang et al., 2025; Lee et al., 2026). These methods improve an agent without changing its underlying model.

A central challenge is how to direct this evolution. At each round, the optimizer must decide which unresolved improvement direction to address, which candidate to continue from, and what change to make. Existing methods use prior harnesses and execution traces (Hu et al., 2025; Lee et al., 2026), reflective updates (Agrawal et al., 2026; Shinn et al., 2023; Madaan et al., 2023), or structured memory (Ouyang et al., 2026; Xu et al., 2025; Zhang et al., 2025a) to inform subsequent candidate changes, but the directions that should guide subsequent evolution typically remain implicit. This can become problematic when the search revisits an earlier branch or switches to a different parent: previously identified directions may be forgotten or rediscovered, and evidence gathered on one branch may fail to inform later evolution on another. At the same time, each attempted evolution produces new evidence about what remains unresolved, what has been addressed, and what new directions have emerged. Harness evolution thus both depends on and produces guidance for subsequent evolution.

We therefore introduce *RelEvolve*, which separates executable candidate lineage from a persistent search state of unresolved improvement directions. We call each direction an *inquiry*. Open inquiries guide which candidate to continue from and what change to make, while each evaluation updates

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.

the inquiries that guide later rounds. At each round, the *Curator* updates the inquiry state and selects a parent; the *Evolver* then uses the selected parent and the currently open inquiries to generate a child. RelEvolve realizes this loop through three linked structures: the *Harness Lineage* records executable inheritance, the *Inquiry Graph* maintains the inquiry state across branches, and the *Relational Trace* links the inquiries that guided each evolution to the corresponding harness change and evaluation evidence. In this way, inquiries guide where the search continues and what it changes, while evaluation evidence updates the inquiries that guide later rounds.

We evaluate RelEvolve across five benchmarks spanning multi-hop question answering (Yang et al., 2018), planning with PlanBench (Valmeekam et al., 2023), code generation with LiveCodeBench (Jain et al., 2025), interactive SQL with InterCode-SQL (Yang et al., 2023), and browser interaction with MiniWoB++ in BrowserGym (Liu et al., 2018; de Chezelles et al., 2025), using two LLM backbones. RelEvolve improves held-out test performance by 4.4% absolute over the strongest evaluated baseline in each setting and by 12.5% absolute over the unoptimized harness, averaged across settings, and three independent search seeds on HotpotQA and InterCode-SQL preserve seed-level advantages of 3.37% and 2.66% absolute over the strongest alternative in these two representative settings. These comparisons share a candidate-harness validation budget. Our contributions are threefold: (1) we formulate harness optimization as a coupled process in which inquiries guide harness evolution and the resulting evidence updates the inquiries that guide later rounds; (2) we realize this coupling with a Harness Lineage, Inquiry Graph, and Relational Trace that separate executable history from inquiry state while linking each attempted change to the inquiries it targets and the evidence it produces; and (3) we evaluate this design across five benchmarks and two backbones, complemented by seed-sensitivity and ablation analyses.

2 Related Work

Executable agent design. Automated agent-design methods search over the surrounding program. ADAS searches over complete agent designs (Hu et al., 2025), AFlow (Zhang et al., 2025b) and A^{2}Flow (Zhao et al., 2026) automate workflow generation, and AgentSquare (Shang et al., 2025) searches a modular design space. STOP, the Darwin Gödel Machine, ShinkaEvolve, and FunSearch further study recursive improvement or evolutionary discovery over executable agents or programs (Zelikman et al., 2024; Zhang et al., 2026a; Lange et al., 2026; Romera-Paredes et al., 2024). ADAS and Meta-Harness are the closest full-harness search baselines considered here. Meta-Harness is especially close: its proposer inspects prior harnesses, scores, and execution traces to diagnose behavior and propose new harnesses (Lee et al., 2026). These methods optimize the executable agent itself. RelEvolve instead maintains a persistent optimizer-side inquiry state alongside executable candidate lineage, so unresolved directions can survive branch switches and inform later selection and mutation.

Agent self-improvement from feedback. Instruction optimizers search textual program components and demonstrations. MIPROv2 optimizes instructions and demonstrations for multi-stage programs, DSPy compiles declarative language-model calls into optimized pipelines, and OPRO casts language models as optimizers (Opsahl-Ong et al., 2024; Khattab et al., 2024; Yang et al., 2024). TextGrad propagates natural-language feedback through compound systems, while GEPA combines reflective mutation with Pareto-based candidate selection (Yuksekgonul et al., 2025; Agrawal et al., 2026). Reflexion and Self-Refine use verbal feedback for subsequent attempts or refinements (Shinn et al., 2023; Madaan et al., 2023). RelEvolve applies feedback-driven refinement to executable-harness search: evaluation evidence is organized as inquiries that guide later parent selection and candidate mutation.

Evolving memory. MemoryBank stores long-term information, while ReasoningBank distills experience into reasoning memories for later use (Zhong et al., 2024; Ouyang et al., 2026). A-Mem organizes experiences as linked, evolving memory entries; G-Memory traces multi-agent experience with a hierarchical memory structure (Xu et al., 2025; Zhang et al., 2025a). Whereas these methods organize experience for the agent to use at runtime, RelEvolve organizes optimizer-side search experience as evidence-linked inquiries that are updated by harness-evolution evidence and guide the Curator and Evolver.

Structured search and refinement. Structured search methods organize reasoning or interaction as trees or graphs (Yao et al., 2023; Zhou et al., 2024; Besta et al., 2024). DoVer and Failure-Driven Workflow Refinement use observed failures to diagnose or revise agent procedures (Ma et al., 2026; Zhang et al., 2026b). Their structured representations center on reasoning traces, workflow revisions, or executable candidates. RelEvolve instead structures the optimizer-side search state: the Inquiry Graph records how improvement directions develop, and the Relational Trace connects those directions to the decisions they guide.

3 METHOD

3.1 RELEVOLVE OVERVIEW

RelEvolve optimizes an executable agent harness H under a per-benchmark budget on candidate-harness validation executions. A harness specifies how the model is prompted, which memory and tools it can access, how intermediate and final outputs are organized, and how control flow proceeds; model parameters remain fixed. To direct this search, RelEvolve separates executable candidate lineage from a persistent search state of unresolved improvement directions. The harness lineage records evaluated harnesses and their executable inheritance, the *Inquiry Graph* tracks improvement directions, their lifecycle status and relations, and their evidence links, and the *Relational Trace* links each targeted inquiry to the resulting harness change and its evaluation evidence. Open inquiries therefore guide both parent selection and candidate mutation. The Inquiry Graph functions as optimizer-side search state: it exposes lifecycle, provenance, and evidence links used by both the Curator and Evolver but is not executed by the task harness. It is shared across candidate branches within one search run (Section 3.3); independent runs and benchmark–backbone settings maintain separate inquiry states.

As shown in Figure 1, the Curator updates the Inquiry Graph and selects a parent from the harness lineage. The Evolver uses that parent and the open inquiries to generate a child. Evaluating the child extends the harness lineage and Relational Trace, so the resulting evaluation evidence updates the inquiry state. Algorithm 1 specifies the resulting search and final selection: the selected harness is the one with the highest validation score, with ties broken by the earlier harness, and is subsequently evaluated under the test protocol in Section 4.1.

3.2 CURATING THE INQUIRY FRONTIER

An *inquiry* is an optimization-scoped record of a candidate improvement direction, such as a behavioral issue, diagnosis, or opportunity; it is evidence-linked rather than a generic memory entry, and its status indicates whether the concern remains open. It belongs to the Inquiry Graph for the current search run and, while open, supplies inquiry guidance for subsequent evolution.

The Curator updates the Inquiry Graph through four transitions. It may create a new inquiry for a newly observed issue, revise an existing inquiry as evidence sharpens it, split a broad inquiry into more specific directions, or close an inquiry when recorded evidence supports resolving its concern. Each transition records the evidence supporting the change; revision updates scope while preserving identity, and split relates child inquiries to their source. Targeting an inquiry states the intent of a proposed change. Closed inquiries remain in the search history and can seed a linked new inquiry if the concern recurs.

Let the Inquiry Graph be $\mathcal{G}_t = (V_{I,t}, \mathcal{C}_{I,t})$, where $V_{I,t}$ is the set of inquiry nodes and $\mathcal{C}_{I,t}$ is the set of typed inquiry edges recording lifecycle or semantic relations. The subscript I denotes the Inquiry Graph, and t denotes the search round. The Curator produces the updated graph $\mathcal{G}_t^+ = (V_{I,t}^+, \mathcal{C}_{I,t}^+)$. The *Inquiry Frontier* used in round t consists of its currently open inquiries:

$$\mathcal{F}_t = \mathrm{Active}(\mathcal{G}_t^+) = \{I \in V_{I,t}^+ : z_t^+(I) = \mathrm{open}\}. \tag{1}$$

Here, $z_t^+(I)$ denotes the inquiry's lifecycle status after curation. The Frontier contains the available open inquiries; an individual evolution may target a subset of them. Closed inquiries remain in the Graph as part of the search history.

The Curator interprets the latest harness evidence in the context of the harness lineage, Inquiry Graph, and Relational Trace. Evidence associated with an existing inquiry updates that inquiry;

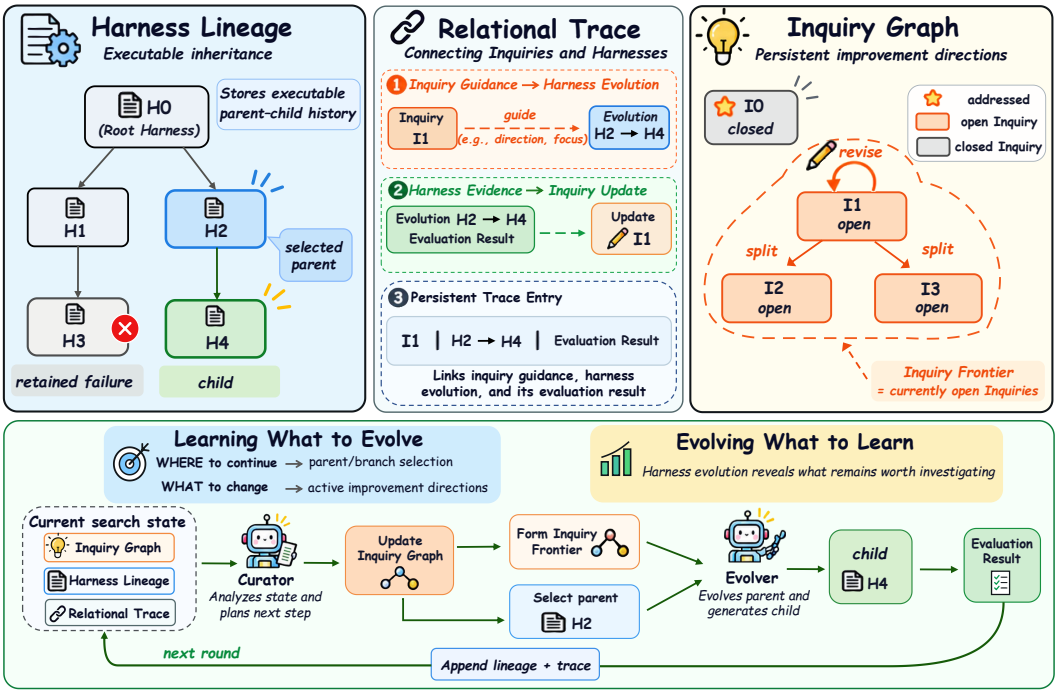


Figure 1: Overview of RelEvolve. The Curator updates the Inquiry Graph, closes addressed inquiries, and selects a parent from the Harness Lineage. The Evolver uses the selected parent and Inquiry Frontier to generate a child. The child's harness evidence is linked to the targeted inquiries through the Relational Trace.

a distinct behavioral issue motivates a new inquiry. If behavior corresponding to a closed inquiry reappears on another branch, the Curator creates a linked new inquiry.

Revision preserves the inquiry's identity while updating its current state; splitting records the relation between a broader inquiry and its more specific directions. These relations describe inquiry-level context; executable inheritance remains in the harness lineage, and the broader inquiry can remain open. In Figure 1, I_1 remains open alongside I_2 and I_3, whereas I_0 has been closed.

3.3 Inquiry Guidance for Harness Evolution

The *Harness Lineage* $\mathcal{L}_t = (V_{H,t}, \mathcal{C}_{H,t})$ contains one node for each evaluated harness in $V_{H,t}$ and one parent–child edge for each evolution in $\mathcal{C}_{H,t}$. Selecting a parent determines *where to continue*: search may extend the current branch or revisit an earlier harness. Unsuccessful children remain in the harness lineage as records of attempted changes, as illustrated by H_3 in Figure 1. The Curator selects the next parent from the evaluated harness lineage using the updated Frontier and Relational Trace. The Evolver then receives the selected parent, the Inquiry Frontier, and relevant historical context to decide *what to change*; it generates an executable child and records the intended behavioral change. An evolution may combine compatible directions or leave some open inquiries untargeted.

The *Harness Evidence* consists of the attempted harness evolution, its validation score, and target-blind feedback. Validation scores support parent selection and final harness selection, whereas target-blind feedback supports diagnosis and candidate mutation. The Curator receives the harness lineage, Inquiry Graph, Relational Trace, and latest harness evidence; it may update the graph and select one eligible evaluated parent, but it cannot edit harness source or select an individual inquiry. The Evolver receives the selected parent, all open inquiries, the parent edge and sibling history, and permitted target-blind feedback. It may generate an executable child harness, but must preserve the public harness interface and may not use hidden labels or case-specific rules.

Harness changes may affect prompting, memory and evidence handling, tool orchestration, and control flow within the task interface; model weights, the evaluator, data partitions, and permitted resources remain fixed. Benchmark-specific interfaces and resource limits belong to the experimental protocol in Appendix A, and the full system prompts and submission procedures are given in Appendix A.9.

Within a search run, inquiry guidance is shared across branches: selecting a different parent does not reset the inquiry state. The Relational Trace makes earlier harness changes and their evaluation evidence available when considering another change, whereas executable inheritance connects each child only to its selected parent. In Figure 1, the Curator selects H_2 as the parent, and inquiry I_1 guides the evolution $H_2 \to H_4$. The Evolver records the behavioral change a_t and the inquiry set $\mathcal{I}_{a_t} \subseteq \mathcal{F}_t$ used for that attempt. The Trace preserves the joint association between these inquiries, the attempted change, and the subsequent evidence; it does not assign a score change to individual inquiries. Draft evaluations, when enabled, provide diagnostic feedback only.

Algorithm 1 RelEvolve: Harness evolution with inquiry updating

Input: root harness H_0, feedback example set D_{fb}, validation example set D_{val}, number of evolution rounds B.
Output: selected harness $\widehat{H}$.

$\mathcal{L}_0 \leftarrow (\{H_0\}, \emptyset)$; $\mathcal{T}_0 \leftarrow \emptyset$
$R_0 \leftarrow \textsc{Evaluate}(H_0, D_{\text{val}}, D_{\text{fb}})$; $E_0 \leftarrow (\emptyset, R_0)$
$\mathcal{G}_0 \leftarrow (\emptyset, \emptyset)$
for $t = 0, \ldots, B-1$ **do**
  $(\mathcal{G}_t^+, H_{p_t}) \leftarrow \textsc{Curator}(\mathcal{L}_t, \mathcal{G}_t, \mathcal{T}_t, E_t)$
  $\mathcal{F}_t \leftarrow \text{Active}(\mathcal{G}_t^+)$
  $(H_{c_t}, a_t, \mathcal{I}_{a_t}) \leftarrow \textsc{Evolver}(H_{p_t}, \mathcal{L}_t, \mathcal{F}_t, \mathcal{T}_t, D_{\text{fb}})$
  $R_{t+1} \leftarrow \textsc{Evaluate}(H_{c_t}, D_{\text{val}}, D_{\text{fb}})$
  Form $\mathcal{L}_{t+1}$ by adding H_{c_t} and an evidence-linked edge $H_{p_t} \to H_{c_t}$ carrying a_t and R_{t+1}
  $e_{t+1} \leftarrow (H_{p_t}, a_t, H_{c_t})$; $E_{t+1} \leftarrow (e_{t+1}, R_{t+1})$
  $\tau_{t+1} \leftarrow (\mathcal{I}_{a_t}, E_{t+1})$
  $\mathcal{T}_{t+1} \leftarrow \mathcal{T}_t \cup \{\tau_{t+1}\}$; $\mathcal{G}_{t+1} \leftarrow \mathcal{G}_t^+$
end for
$\widehat{H} \leftarrow \arg\max_{H \in V_{H,B}} \text{val}(H)$, where $\text{val}(H)$ is the validation score; freeze and return $\widehat{H}$, with ties broken by the earlier harness

3.4 The Inquiry–Harness Evolution Loop

Evaluating the child provides the *evaluation result* $R_{t+1} = (O_{t+1}, \Delta_{t+1})$, where O_{t+1} is the validation score and Δ_{t+1} is the target-blind feedback record from the feedback set. The *Harness Evidence* combines this result with the attempted harness evolution $e_{t+1} = (H_{p_t}, a_t, H_{c_t})$:

$$E_{t+1} = (e_{t+1}, R_{t+1}). \tag{2}$$

The validation score supports harness selection, while the Curator uses E_{t+1} to update the inquiry state in the next round.

A persistent trace entry links the targeted inquiries to their corresponding harness evidence:

$$\tau_{t+1} = (\mathcal{I}_{a_t}, E_{t+1}). \tag{3}$$

The Relational Trace $\mathcal{T}_t$ contains the entries recorded through round $t-1$. It records inquiry references, harness-edge references, and paired evaluation results. Inquiry references identify the guidance used for an evolution, harness-edge references identify the attempted change, and the paired evaluation result records the outcome. When an evolution targets several inquiries, its evidence is stored jointly.

Update order. The first Curator call bootstraps the inquiries from the root harness evidence E_0. Each subsequent round updates the Inquiry Graph, selects a parent, generates one child, and records its harness evidence. The final child is retained in the search history even though no later curation occurs; the selected harness is frozen before test evaluation. The Curator's diagnosis remains a hypothesis about the behavior under investigation; later evidence determines whether it is revised, split, or closed.

4 EXPERIMENTS

We evaluate RelEvolve on three axes: breadth across five benchmarks and two LLM backbones, mechanism analysis through information-flow and inquiry-state representation ablations on HotpotQA, and seed sensitivity on two representative task–backbone settings. Table 1 reports cross-task breadth; Tables 2 and 3 present ablations; Table 4 reports seed-level variation, including the Rolling Flat Summary ablation also evaluated in Table 3.

4.1 EXPERIMENTAL SETUP

The task suite covers multi-hop question answering with HotpotQA (Yang et al., 2018), planning with PlanBench (Valmeekam et al., 2023), code generation with LiveCodeBench (Jain et al., 2025), interactive SQL with InterCode-SQL (Yang et al., 2023), and browser interaction with MiniWoB++ in BrowserGym (Liu et al., 2018; de Chezelles et al., 2025). Their primary metrics are F1, success rate, pass@1, IoU, and success rate, respectively. We evaluate GPT-5.6 Luna and GLM-5.3-Flash separately. The same backbone model powers the harness solver, Curator, and Evolver in each model setting. The baselines comprise the Root Harness as the unoptimized starting point, MIPROv2 and GEPA as prompt-component optimizers (Opsahl-Ong et al., 2024; Agrawal et al., 2026), and ADAS and Meta-Harness as full-harness search methods (Hu et al., 2025; Lee et al., 2026). RelEvolve is also a full-harness search method and uses ten evolution rounds. Appendix A.1 describes data splits and the evaluation protocol; Appendix A.2 describes baseline adaptation.

All optimized methods share the same feedback, validation, and held-out test splits, backbone model, evaluation protocol, and per-benchmark cap on candidate-harness validation executions. Validation scores guide parent selection and final harness selection, whereas the held-out test split is not used during either. The cap controls evaluation cost but not optimizer-side compute, such as LLM calls, tokens, generated candidates, or diagnostic calls. Feedback and optional draft-feedback episodes are bounded separately. Figure 2(b) reports cumulative model-token usage for HotpotQA. Behavioral feedback does not expose reference answers or test examples, although scalar scores may depend on the evaluation metric. Because full-harness search is computationally expensive, we report one independent search run per method and setting, as in ADAS and Meta-Harness (Hu et al., 2025; Lee et al., 2026); Section 4.4 complements this breadth evaluation with a three-seed sensitivity analysis on two representative task–backbone settings. A static exact and near-exact leakage audit did not detect literal test leakage (Appendix A.8).

4.2 MAIN RESULTS

Across five benchmarks and two LLM backbones, RelEvolve outperforms the strongest evaluated baseline in each setting by 4.4 percentage points on average and the unoptimized harness by 12.5 percentage points on average; the strongest baseline varies by setting, and gaps relative to it range from 2.00 to 10.00 percentage points, with the largest gap on LiveCodeBench and the smallest on PlanBench. MIPROv2 and GEPA optimize prompt components while leaving executable control flow fixed; ADAS, Meta-Harness, and RelEvolve search full harnesses, making ADAS and Meta-Harness the most direct comparisons. Section 4.4 reports seed sensitivity on two representative settings.

4.3 ABLATIONS

All ablations use HotpotQA with GPT-5.6 Luna, ten evolution rounds, the same 150/150/300 feedback/validation/test splits, and the same candidate-harness validation-execution cap. The primary metric is held-out test F1. Information-flow ablations disable one inquiry pathway at runtime; representation ablations change how improvement information is retained across rounds. Each is evaluated as an independent search.

4.3.1 INFORMATION-FLOW ABLATIONS

Table 2 removes one information flow in the search loop while retaining the other two; the *w/o Inquiry* ablation removes all three. In the reported HotpotQA search, preventing inquiry updating from incorporating new evidence lowers F1 from 89.84 to 86.69; removing inquiry guidance from

Table 1: Held-out test performance (%) on multi-hop QA, planning, code generation, interactive SQL, and browser interaction with GPT-5.6 Luna and GLM-5.3-Flash. Optimized methods report one independent search run per setting; Section 4.4 reports seed sensitivity for two representative settings. Bold marks the best result in each column. † denotes prompt-component optimizers, ‡ full-harness search methods, and the Root Harness is the unoptimized starting point. Appendix A.1 gives protocol details.

Method	HotpotQA	PlanBench	LiveCodeBench	InterCode-SQL	MiniWoB++
GPT-5.6 Luna					
Root Harness	74.71	54.00	58.67	46.21	72.50
MIPROv2†	82.93	70.00	60.67	47.23	76.67
GEPA†	85.51	70.00	64.00	47.72	74.17
ADAS‡	81.08	68.67	63.33	50.83	75.83
Meta-Harness‡	85.84	64.67	62.67	51.99	75.00
RelEvolve‡	**89.84**	**72.67**	**70.67**	**55.05**	**82.50**
GLM-5.3-Flash					
Root Harness	71.48	70.00	46.67	51.33	75.83
MIPROv2†	86.84	75.33	53.33	53.51	76.67
GEPA†	87.07	80.00	46.67	52.45	75.83
ADAS‡	87.89	74.00	54.67	52.31	78.33
Meta-Harness‡	86.65	78.00	50.00	53.69	77.50
RelEvolve‡	**91.24**	**82.00**	**64.67**	**56.50**	**81.67**

Table 2: Information-flow ablations on HotpotQA with GPT-5.6 Luna. Values are held-out test F1 from one independent search run.

Setting	F1
RelEvolve	89.84
w/o Inquiry update	86.69
w/o Inquiry	83.89
w/o Inquiry-guided parent selection	86.81
w/o Inquiry-guided candidate mutation	84.49

Table 3: Inquiry-state representation ablations on HotpotQA with GPT-5.6 Luna. Values are held-out test F1 from one independent search run.

Setting	F1
RelEvolve	89.84
Persistent Inquiry List	87.17
w/o Inquiry Persistence	85.97
w/o Inquiry–Evidence Links	85.31
Rolling Flat Summary	85.19

parent selection lowers it to 86.81; removing it from candidate mutation lowers it to 84.49; and removing all inquiry pathways entirely yields 83.89. These results are consistent with each evaluated information flow—updating, parent selection, and candidate mutation—contributing to the coupled design. Appendix A.5.1 specifies the information available to each role in every condition.

4.3.2 Inquiry-State Representation Ablations

Table 3 compares the full inquiry state with alternative ways of retaining improvement information under the same inquiry context budget. RelEvolve obtains 89.84 F1, compared with 87.17 for Persistent Inquiry List, 85.97 without persistence, 85.31 without inquiry–evidence links, and 85.19 for Rolling Flat Summary. These results support an advantage for persistent, evidence-linked inquiry state over the simpler representations evaluated here on HotpotQA under the same context budget. They do not isolate individual graph relations. Appendix A.5.1 defines each representation.

4.4 Seed Sensitivity

For HotpotQA with GPT-5.6 Luna and InterCode-SQL with GLM-5.3-Flash, we evaluate three independent search seeds for RelEvolve, Rolling Flat Summary, and Meta-Harness; seed 1 corresponds to Table 1 for RelEvolve and Meta-Harness. For Rolling Flat Summary, seed 1 corresponds to Table 3 on HotpotQA; the InterCode-SQL seed 1 is reported in the appendix seed-level table. Each seed consists of an independent search, validation-based harness selection, and held-out test evaluation. These settings span question answering and interactive SQL across two backbones; the SDs

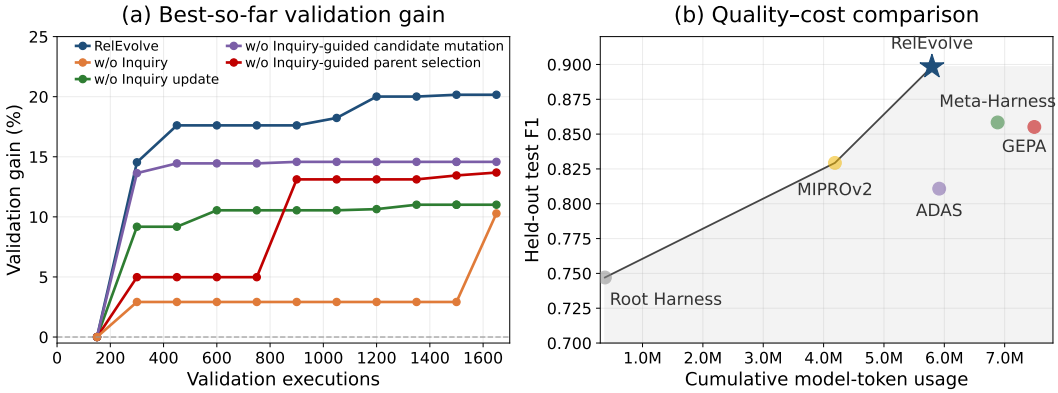


Figure 2: Ablation dynamics and quality–cost comparison on HotpotQA. (a) Best-so-far validation gain over the Root Harness for each information-flow condition; the x axis counts cumulative candidate-harness validation executions. Each condition uses one independent search run. (b) Held-out test F1 versus cumulative model-token usage for selected harnesses.

summarize variation across the full search pipeline in these two representative settings. The Rolling Flat Summary condition is the same representation ablation listed in Table 3. RelEvolve averages 89.48 F1 and 55.90 IoU, exceeding the strongest alternative in each setting by 3.37 and 2.66 percentage points, respectively (Table 4); seed-level scores are in Appendix A.1.

Table 4: Held-out test performance across three independent search seeds. HotpotQA reports F1 with GPT-5.6 Luna; InterCode-SQL reports IoU with GLM-5.3-Flash. Values are mean ± SD. Seed-level scores are in Appendix A.1.

	Benchmark	
Method	HotpotQA	InterCode-SQL
RelEvolve	89.48 ± 0.34	55.90 ± 0.66
Rolling Flat Summary	86.11 ± 0.87	52.77 ± 0.43
Meta-Harness	85.27 ± 0.62	53.24 ± 0.81

4.5 Search Dynamics, Cost, and Case Study

On HotpotQA, Figure 2(a) plots the best-so-far validation gain over the Root Harness for one independent search run per information-flow condition; ablation differences emerge during search: RelEvolve reaches the highest value in the reported trajectories, while several ablations plateau at lower levels. Figure 2(b) plots held-out test F1 against cumulative model-token usage across search and final test evaluation (Appendix A.3); RelEvolve attains the highest held-out test F1 while using the fewest cumulative model tokens among the evaluated full-harness search methods. Beyond these aggregate comparisons, Figure 3 traces one HotpotQA search in which an unsuccessful child leaves an inquiry open, later evidence splits it, and the updated Inquiry Frontier informs parent selection and the next harness change; this case study illustrates the coupled loop in one search trajectory.

5 Conclusion

RelEvolve couples harness evolution with inquiry updating: inquiries guide where evolution continues and what it changes, while evaluation evidence updates the inquiries that guide later rounds. By separating candidate lineage from inquiry state, RelEvolve preserves guidance across branches. Across five benchmarks and two LLM backbones, it improves held-out test performance by 4.4 percentage points over the strongest evaluated baseline in each setting and by 12.5 percentage points over the unoptimized harness, averaged across settings. Three independent search seeds on HotpotQA and InterCode-SQL maintain this improvement in these two representative settings. On

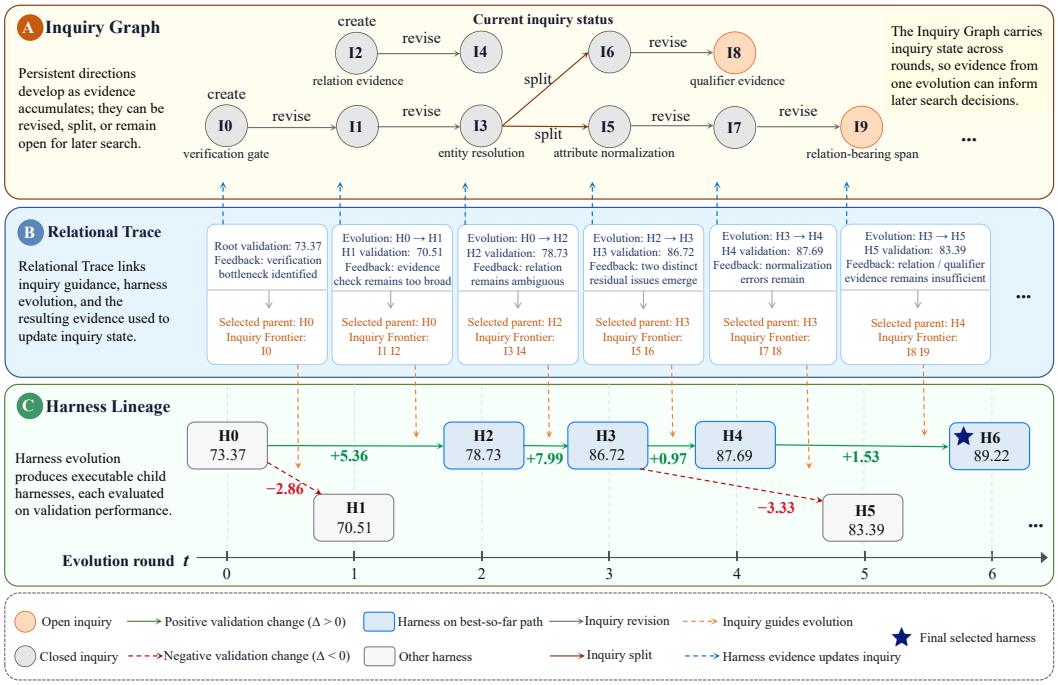


Figure 3: An illustrative HotpotQA search trace. (A) A time-unrolled view of the Inquiry Graph, where I_k denotes inquiry-state snapshots. A revise edge connects successive states of the same inquiry, whereas a split edge introduces a more specific inquiry. (B) Each card summarizes the latest harness evidence. Within each card, the arrow indicates that the harness evidence updates the inquiry state before the resulting parent selection and Inquiry Frontier are shown for the next evolution. (C) Harness Lineage with validation F1 scores (%); edge labels report child-minus-parent score differences in percentage points. Table 1 reports held-out test F1.

HotpotQA, ablations indicate that removing any of the inquiry pathways, persistence, or inquiry–evidence links lowers performance.

AI Use Statement

Generative AI tools were used under author direction to assist with manuscript drafting and auxiliary software editing and debugging. They were not used to formulate the research hypotheses or conceptual framework, design the methodology or experiments, implement the reported method, generate or preprocess datasets, or produce reported measurements or findings. This assistance was separate from the language models evaluated in the paper. The authors reviewed and verified all AI-assisted text and code, and take responsibility for the final content of the submission.

Reproducibility Statement

Section 3.1 and Algorithms 1 and 2 define the search process. Appendix A and the supplementary release provide data partitions, evaluation protocols, baseline configurations, role interfaces, ablation controls, prompts, sampling settings, run lineages, inquiry states, generated harnesses, and validation/test scores. Request identifiers name the configured endpoints used in the reported experiments.

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 *The Fourteenth International Conference on Learning Representations*, 2026.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 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. doi: 10.1609/aaai.v38i16.29720.

Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Massimo Caccia, Alexandre Drouin, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Quentin Cappart, Graham Neubig, Russ Salakhutdinov, and Nicolas Chapados. The browsergym ecosystem for web agent research. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=5298fKGmv3.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025. arXiv:2408.08435.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

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 state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, 2024.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=lKEdGCoDNC.

Yoonho Lee, Roshen Sanjay Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses. In *Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=tmbOUyFx3R.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-sqls. In *Advances in Neural Information Processing Systems 36*, pp. 42330–42357, 2023. doi: 10.52202/075280-1835.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations*, 2018. URL https://arxiv.org/abs/1802.08802.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. DoVer: Intervention-driven auto debugging for LLM multi-agent systems. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=mrEK16Jy6h.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36*, pp. 46534–46594, 2023. doi: 10.52202/075280-2019.

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. doi: 10.18653/v1/2024.emnlp-main.525.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=jL7fwchScm.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6.

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 *The Thirteenth International Conference on Learning Representations*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36*, pp. 8634–8652, 2023. doi: 10.52202/075280-0377.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. PlanBench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Advances in Neural Information Processing Systems 36*, pp. 38975–38987, 2023. doi: 10.52202/075280-1693.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems 38*, pp. 20004–20031, 2025. doi: 10.52202/085713-0593.

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

John Yang, Akshara Prabhakar, Karthik R. Narasimhan, and Shunyu Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. In *Advances in Neural Information Processing Systems 36*, pp. 23826–23854, 2023. doi: 10.52202/075280-1035.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. 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. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

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. In *Advances in Neural Information Processing Systems 36*, pp. 11809–11822, 2023. doi: 10.52202/075280-0517.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639:609–616, 2025. doi: 10.1038/s41586-025-08661-4.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): Recursively self-improving code generation. In *Conference on Language Modeling*, 2024.

Guibin Zhang, Muxin Fu, Kun Wang, Frank Wan, Miao Yu, and Shuicheng Yan. G-Memory: Tracing hierarchical memory for multi-agent systems. In *Advances in Neural Information Processing Systems*, 2025a. URL https://openreview.net/forum?id=mmIAp3cVS0.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *The Fourteenth International Conference on Learning Representations*, 2026a.

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. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Jusheng Zhang, Jing Yang, Kaitong Cai, Ziliang Chen, Yongsen Zheng, Kwok Yan Lam, Liang Lin, and Keze Wang. Failure-driven workflow refinement. In *Proceedings of the Forty-Third International Conference on Machine Learning*, 2026b. URL https://openreview.net/forum?id=GbYHY1RVUa.

Mingming Zhao, Xiaokang Wei, Yuanqi Shao, Kaiwen Zhou, Lin Yang, Siwei Rao, Junhui Zhan, and Zhitang Chen. A^2Flow: Automating agentic workflow generation via self-adaptive abstraction operators. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 29930–29938, 2026. doi: 10.1609/aaai.v40i35.40240.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024. doi: 10.1609/aaai.v38i17.29946. URL https://doi.org/10.1609/aaai.v38i17.29946.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 62138–62160, 2024.

A Implementation and Reproducibility Details

This appendix provides implementation details for data partitions, analysis definitions, inquiry schemas, role interfaces, baseline adaptation, and ablations.

A.1 Benchmark Protocols and Comparison Boundaries

Table 5 gives the per-benchmark cap on validation executions for candidate harnesses shared by all optimized methods. One validation execution is one candidate-harness execution on one validation example for search or selection. RelEvolve evaluates the root harness once and one child in each of its ten evolution rounds, so its cap is $(10 + 1)|D_{\text{val}}|$: 1,650 executions for HotpotQA and 880 for each benchmark with 80 validation examples. These caps count only candidate-harness validation executions; they do not count LLM calls, input/output tokens, generated candidates, or diagnostic calls. The number of candidates can differ by method. Feedback and optional draft-feedback episodes are bounded separately and excluded from this cap, as are held-out test executions. The Feedback column in Table 5 reports the size of the available feedback pool, not the number of feedback episodes consumed. Feedback supports diagnosis, validation supports search and selection, and test examples assess the frozen harness. Behavioral feedback contains no reference answers or test examples, although scalar scores may depend on the evaluation metric.

The table fixes the benchmark families and resource-facing limits. Each RelEvolve search uses ten evolution rounds. HotpotQA uses a 1,000-question HotpotQA train subset with a fullwiki-style retrieval interface over Wikipedia-2017 abstracts and up to five retrieved passages. PlanBench uses the Mystery/Randomized Blocksworld variants with a 50/50 panel balance and the official VAL checker. LiveCodeBench uses the official code-generation hard-only split from `release_v6`, covering contest dates from May 2023 through April 2025, with the functional-correctness checker. The static audit addresses harness-level leakage, not backbone pretraining contamination; we therefore read LiveCodeBench as a same-backbone method comparison on the fixed split. InterCode-SQL uses BIRD (Li et al., 2023) through the official SQL interface with a public-schema summary and up to two SQL actions per case. MiniWoB++ uses BrowserGym with up to twenty environment steps and the official binary success signal. Each optimization round samples up to twelve feedback examples under fixed panel execution seeds.

InterCode-SQL uses the official execution-based IoU reward. For each case, the submitted and gold SQL queries are executed and each returned row is stringified; the score is the summed multiplicity of shared rows divided by the summed maximum row multiplicity. If both result sets are empty, the score is 1. When a non-empty intersection exists, the reward is scaled by the Kendall rank correlation between matched rows. IoU gives partial credit for partially correct result sets; success rate is the fraction of cases whose best reward reaches the official success threshold.

Table 5: Task partitions, primary metrics, and validation-execution caps for candidate harnesses. Feedback, validation, and test report the available examples in each split; the Feedback count is a pool size, not feedback episodes consumed. The cap excludes separately bounded feedback and draft-feedback episodes and held-out test evaluation.

Benchmark	Task type	Metric	Feedback	Validation	Test	Validation Exec. Cap
HotpotQA	Multi-hop QA	F1	150	150	300	1,650
PlanBench	Planning	Success rate	80	80	150	880
LiveCodeBench	Code generation	Pass@1	80	80	150	880
InterCode-SQL	Interactive SQL	IoU	80	80	150	880
MiniWoB++	Browser interaction	Success rate	80	80	120	880

The two RelEvolve HotpotQA evaluations reported in Table 1 each contain 300 examples and use execution seed 1000. The Luna and GLM harnesses were selected after ten evolution rounds under the fixed feedback/validation/test protocol. The task results are test evaluations of the selected frozen harnesses.

Table 6: Seed-level results. Seed 1 for RelEvolve and Meta-Harness corresponds to the matching benchmark–backbone row in Table 1; for Rolling Flat Summary, the HotpotQA seed-1 score matches Table 3, and the InterCode-SQL seed-1 score is reported in this table. Seeds 2 and 3 are additional independent searches followed by held-out test evaluation.

Benchmark / backbone	Metric	Method	Seed 1	Seed 2	Seed 3
HotpotQA / GPT-5.6 Luna	F1	RelEvolve	89.84	89.45	89.16
HotpotQA / GPT-5.6 Luna	F1	Rolling Flat Summary	85.19	86.91	86.22
HotpotQA / GPT-5.6 Luna	F1	Meta-Harness	85.84	85.37	84.61
InterCode-SQL / GLM-5.3-Flash	IoU	RelEvolve	56.50	55.20	56.00
InterCode-SQL / GLM-5.3-Flash	IoU	Rolling Flat Summary	52.75	52.35	53.20
InterCode-SQL / GLM-5.3-Flash	IoU	Meta-Harness	53.69	53.72	52.31

A.2 Baseline Adaptation

MIPROv2 and GEPA optimize prompt components and leave executable control flow fixed. The MIPROv2 adapter bootstraps grounded instruction and demonstration candidates, samples joint instruction–demonstration combinations with multivariate TPE, and scores each candidate on the bounded validation evaluator while retaining the fixed program topology. GEPA reflects on target-blind feedback traces and updates a validation Pareto frontier. ADAS and Meta-Harness search the full harness: ADAS stages and validation-evaluates archive candidates, and Meta-Harness preserves its proposer's authorized filesystem access to prior candidate sources, scores, and target-blind execution traces. The adapters preserve each baseline's candidate-generation and search behavior while adapting only their evaluation-facing interfaces. Raw reference answers and test examples remain unavailable to all search roles.

All baselines use the same feedback, validation, and test splits, backbone model, per-benchmark cap on candidate-harness validation executions, and evaluation protocol. We therefore compare methods under a shared candidate-harness validation-execution budget; Figure 2(b) reports model-token usage. The supplementary release records baseline candidate generation, diagnostic calls, retries, and hyperparameters. Run artifacts include the corresponding search lineages, inquiry states, generated harnesses, optimizer conversations, and validation/test scores.

A.3 Analysis Definitions

Performance is reported as the benchmark's native metric on the frozen test split, expressed as a percentage. Reported gaps are measured on the frozen held-out test split. A reported gap is the absolute percentage-point difference from RelEvolve. Each seed corresponds to an independent search run followed by test evaluation.

Figure 2(b) reports cumulative model-token usage as input and output tokens consumed by all model calls across search and final test evaluation, including optional draft-feedback calls. It excludes cached reuse of the common Root Harness initialization, local metric computation, and non-LLM tool execution.

A.4 Inquiry Schema and Context Assembly

Each inquiry record stores an identifier, text, open/closed status, derivation sources, source-harness association, linked evidence identifiers, and experimental trials. Each trial records the decision and harness-edge references, selected parent, generated child, intervention summary, paired feedback delta, and creation time. Inquiry relations store a relation identifier, operation, source and target inquiry identifiers, evidence references, and rationale. Harness edges record targeted and tested inquiry identifiers, the inquiry-selection rationale and evidence references, the generated-child intervention summary, and the paired feedback delta. Inquiry trials, harness edges, and trace entries store different relational references; they do not duplicate the underlying harness or evaluation state.

Context assembly begins with a system prompt and a structured current-round user payload. The Curator payload contains the Harness Lineage index, Inquiry Graph with recent relations, Relational Trace with the focus edge and its latest evidence, latest and best-so-far harness identifiers, and

remaining rounds; read-only tools provide bounded access to inquiries, harnesses, lineage edges, and feedback. The Evolver payload contains the selected parent, all open inquiries, prior child edges, selection rationale and evidence references, incoming parent edge, and feedback budget. If session history exceeds the configured context budget, the runtime retains the system prompt and current user payload, selects the most recent interaction blocks, and replaces omitted blocks with a compact summary. This projection bounds the model-visible context without altering the immutable session ledger.

A.5 Curator, Evolver, and Ablation Protocol

The Curator receives the Harness Lineage, Inquiry Graph, Relational Trace, and latest Harness Evidence $E = (e, R)$, where e is the attempted harness evolution and $R = (O, \Delta)$ is the evaluation result with validation score O and target-blind feedback record Δ. The Evolver receives the selected parent, Inquiry Frontier, and relevant historical context. Neither role receives reference answers or test examples.

In each backbone setting, the configured request model is shared by the harness solver, Curator, and Evolver: GPT-5.6 Luna (request identifier `gpt-5.6-luna`) for the Luna setting and GLM-5.3-Flash (request identifier `glm-5.3-flash`) for the GLM setting. The solver uses temperature 0.0, and the Curator and Evolver use temperature 0.2. These identifiers name the configured endpoints used in the reported experiments.

Curator procedure. Algorithm 2 assesses existing inquiries before creating new directions. In line 3, $\mathrm{Active}(\mathcal{G})$ denotes the open inquiries before this update; line 4 applies the judged transitions in $\mathcal{G}^+$. Its first call applies the procedure to an empty Graph using the root's harness evidence E_0. Parent selection uses the resulting Frontier and the available harness history.

Algorithm 2 Curator update and parent selection

Input: Harness Lineage $\mathcal{L}$, Inquiry Graph $\mathcal{G}$, Relational Trace $\mathcal{T}$, Harness Evidence E.

1 $\mathcal{G}^+ \leftarrow \mathcal{G}$
2 Interpret E using the current inquiries and their Relational Trace
3 **for each** $I \in \mathrm{Active}(\mathcal{G})$ **do**
4 If evidence supports judging I addressed, close it in $\mathcal{G}^+$; otherwise retain, revise, or split the inquiry
5 **end for**
6 Create open inquiries for new or branch-recurrent directions not covered by open inquiries, possibly linked to an earlier closed inquiry
7 Relate new inquiries to the inquiries from which they arose
8 Select H_p from $\mathcal{L}$ using $\mathrm{Active}(\mathcal{G}^+)$ and $\mathcal{T}$
9 **return** $(\mathcal{G}^+, H_p)$

Evolver procedure. The Evolver generates an executable child, a behavioral change summary a_t, and the targeted inquiry set $\mathcal{I}_{a_t}$. Targeting states an intent to investigate; the next Curator update judges whether the evidence supports marking the inquiry addressed.

Lifecycle and historical guidance. Revision advances an inquiry's current state while preserving its identity; splitting introduces more specific directions. Closing records that an inquiry is inactive under its current scope, while closed inquiries remain in the graph and can be linked to a new inquiry if the concern recurs. Trace references retain the guidance used at the time of an evolution. A later revision therefore changes future guidance without rewriting the meaning of the recorded attempt.

Joint targeting and evidence linkage. When a child targets multiple inquiries, the child-level evaluation updates all targeted inquiries as joint evidence. The Curator uses Relational Trace associations to revise the scope of an inquiry and to decide whether a recurrent concern warrants a linked new inquiry. The Trace preserves joint evidence; it does not assign score changes to individual inquiries.

A.5.1 Ablation Input Restrictions

Tables 2 and 3 use HotpotQA with GPT-5.6 Luna, ten evolution rounds, and feedback/validation/test sets of 150/150/300 examples. The cap on validation executions for candidate harnesses is 1,650 executions, as defined in Table 5. Each ablation reports one independent search; the RelEvolve reference is shared across the two tables and corresponds to the score in Table 1. The representation ablations use the same inquiry context budget.

Information-flow ablations differ in which inquiry-guidance path is available to each search role. Representation ablations change how improvement information is retained across rounds.

Information-flow ablations. The *w/o Inquiry update* condition retains the initial inquiries and execution feedback but prevents later evidence from changing the Inquiry Graph. The *w/o Inquiry* condition retains harness lineage and feedback but creates, persists, and exposes no inquiries to either search role. The *w/o Inquiry-guided parent selection* condition retains inquiry updating and candidate-mutation guidance while restricting the parent-selection interface to the harness lineage and task evidence available at that decision point. The *w/o Inquiry-guided candidate mutation* condition applies the corresponding restriction to the Evolver while preserving inquiry-guided parent selection. Each restriction is enforced at runtime, and each condition is an independent search with role-specific context assembly, so the restricted role cannot access inquiry state through tools, histories, or prior calls. These ablations compare the two decision pathways through which inquiry state enters parent selection and candidate mutation.

Inquiry-state representation ablations. *w/o Inquiry Persistence* cannot retrieve prior-round inquiries through histories or tools; the harness lineage remains available. *Rolling Flat Summary* carries improvement information between rounds in one free-text summary, exposed as the inquiry-facing state instead of as individually represented inquiries and typed relations. It uses the same inquiry context budget as the other representation ablations and is also included in the seed sensitivity analysis in Table 6. *Persistent Inquiry List* exposes persistent inquiry text, status, and trial counts without typed relations or inquiry–harness associations. *w/o Inquiry–Evidence Links* exposes inquiry text and harness history without inquiry–modification–result links. These ablations compare alternative forms of retaining improvement information; the conditions are not arranged as a one-component-at-a-time progression.

RelEvolve is shared between Tables 2 and 3. Each table reports test performance, and each ablation is an independent search under the stated benchmark protocol.

A.6 Prompt Templates and Role Interfaces

Complete system prompts and submission procedures are shown in Appendix A.9; the rest of this subsection summarizes the inputs, tools, and output interfaces available to each optimizer role.

Curator inputs and tools. The Curator input contains the Harness Lineage, Inquiry Graph, Relational Trace, the focus edge and its latest evidence, the latest and best-so-far harness identifiers, and the remaining round budget. Read-only tools expose one inquiry, harness, lineage edge, feedback index, or selected feedback cases. The role then calls `submit_inquiry_update` exactly once with zero or more updates, followed by `submit_selection` exactly once. The update operation is one of `create`, `revise`, `split`, or `close`; evidence references must point to persisted lineage or feedback records, not individual case identifiers. Selection is restricted by the runtime to an eligible evaluated parent.

Evolver inputs and tools. The Evolver input contains the selected parent and validation score, prior child edges, all open inquiries, the selection rationale and evidence references, the selected parent's incoming edge, and the feedback budget. The role may use `write_draft`, read permitted feedback cases, read authorized lineage edges or harnesses, and call `evaluate_feedback`. The latter accepts either explicit case IDs or a sample size, but always resolves to the selected parent's fixed feedback set. The terminal `submit_child` records the complete source, the tested open inquiry IDs, and a change summary; the runtime checks that the IDs are open and that the child remains executable.

A.7 LIMITATIONS

RelEvolve studies black-box harness evolution with fixed model weights and benchmark-level budgets. The breadth evaluation matches candidate-harness validation executions but not optimizer-side compute, and seed sensitivity covers only two representative settings. The breadth evaluation uses ten evolution rounds and two API backbones; longer horizons, open-weight models, cross-model roles, and cross-task inquiry reuse may alter trade-offs. Because the Curator and Evolver remain LLM-mediated, RelEvolve is best understood as a persistent-state and information-flow design rather than a new LLM policy, and reported gains may reflect the joint effect of inquiry content, parent selection, and branch revisitation rather than individual graph relations. The case study in Figure 3 is illustrative, and LiveCodeBench is read as a same-backbone comparison on its fixed split rather than a post-training generalization result.

A.8 LEAKAGE AUDIT

We performed a static exact and near-exact leakage audit of the harness with the highest validation score for each method and setting against the feedback and test inputs. The audit checked for exact input strings, case identifiers, submitted outputs, reference-answer literals, and long contiguous n-gram overlap. It did not detect exact matches or long overlaps. This audit addresses harness-level leakage, not backbone pretraining contamination; all compared methods use the same backbone in each setting.

A.9 FULL SYSTEM PROMPTS

Curator system prompt

```
You are the Curator for one evolution round. First, inspect the newest
evaluated child, the Harness Lineage, and the Inquiry Graph, and update the Inquiry Graph when the
evidence supports it. Then select one evaluated parent Harness from which the next child will start.
Do not edit Harness source and do not select inquiries; the Evolver sees every open inquiry as
guidance and later reports which inquiries its child actually tested.

The Inquiry Graph is a persistent search-control state, not an executable controller. An inquiry
records why a direction was proposed, what evidence grounded it, and which children have tested it.
Maintain the graph only when it improves that memory: create a direction, close one that is resolved
or no longer useful, revise an existing direction by advancing it to a new state when its scope changes, or
 split a broad
direction into more specific directions while retaining the broader direction. Submit an empty update
when no change is needed. A failed child falsifies that specific implementation under that evaluation, not
the whole behavioral family; close a direction only under comparable evidence across materially
different implementations.

Choose the parent whose continuation is most likely to improve the validation score, usually the best-so-
far harness,
and briefly state the behavior to retain and the failed direction to avoid. Feedback is target-blind
diagnostic evidence for how behavior changed on specific cases; validation scores are the selection signal
recorded by the system. Do not write case IDs, submitted answers, or per-case rules into the inquiry
text or rationale.

Keep these boundaries: feedback is target-blind, raw validation/test examples and hidden labels are
unavailable, and the selected parent must remain an executable Harness. The system records the
decision and graph history; do not reproduce records already present in the context or tools. Submit
your graph update (possibly empty), then one selection when ready.
```

Evolver system prompt

```
You are the Evolver for one selected experiment. Starting from the
selected parent source, create a complete executable child Harness that you believe improves validation
performance on the task. You may add, remove, rewrite, consolidate, or combine behavior anywhere in
the parent, including the interaction loop, solver-facing task instructions, and message assembly.
Only the Harness public interface (class name, method signatures, and permitted tool access) must
remain unchanged. Prefer rewriting or consolidating overlapping parent guidance over appending
another parallel instruction block.

Every open inquiry is historical induced guidance, not a restriction. Read the open inquiries to
understand what past experiments proposed and tested, then design a genuinely new or improved
mechanism when the evidence suggests it. Inquiry variants and case details are diagnostic evidence,
not implementation requirements. Use authorized target-blind feedback when it helps diagnose
behavior.

When you actively evaluate a draft, use the exact feedback case IDs you diagnosed (or sample from
the parent's fixed feedback set); never validate the edit on a fresh unrelated case set,
because these before/after scores diagnose the change on the exact cases you examined. This
is diagnostic evidence, not selection.
```

```
Keep the Harness executable, preserve its permitted interface and reachable completion path, and do
not encode case-specific rules or hidden information. You may design the solver-facing task
instructions and message assembly directly in the Harness; keep them general to the task and
case-agnostic, consistent with the permitted tools. The prior_child_edges list children that
already exist in the Harness Lineage; open inquiries come from the Inquiry Graph. Use these and
permitted target-blind feedback for diagnosis.
On submission, list in tested_inquiry_ids the open inquiries this child actually tests, leaving it
empty when no inquiry applies. Submit the complete child source with a concise summary of what
changed.
```