From Harness Optimization to Recursive Self-Improvement: A Survey of Language Model Agents

Xin Yu*,† Chengyuan Liu* Wenxin Long* Haochen Zhang*
Lingzhou Xue‡
Department of Statistics, The Pennsylvania State University
University Park, PA 16802, USA

*Equal contribution. †Project Lead.
‡Corresponding author: lzxue@psu.edu

Homepage GitHub

Preprint

Abstract

Language-model agents rely on harnesses that govern tool use, context, memory, and execution. Understanding automated harness revision requires connecting improvement methods to the reusable functionality they produce. This survey examines three linked perspectives: improvement workflows, recursive scope, and concrete modifications across task domains. We compare search strategies, memory management, edit guidance, and integration with model training, and use an existing five-level framework to distinguish harness optimization from recursive self-improvement. Across coding, reasoning and information processing, interactive and embodied tasks, and visual tasks, we trace reported changes from starting capabilities to outcomes. Our synthesis identifies repairs at model–environment interfaces, cases where added tools and context fail to help, and conditions constraining recovery reuse. Cross-domain analogies motivate, rather than establish, transfer. We distinguish task-performance gains from improved subsequent optimization and identify priorities in reliable evaluation, attribution, transfer, and safety.

Keywords: large language model agents; agent harness; harness optimization; recursive self-improvement; self-evolving agents; survey.

1 Introduction

Large language model (LLM) agents act through an *agent harness*: instructions, tool interfaces, context and memory mechanisms, and execution control. Surveys of self-evolving agents and experience-driven adaptation situate these components within a broader landscape of agent improvement (Gao et al., 2026; Jiang et al., 2026a). Interface design affects coding performance (Yang et al., 2024). Harness evolution offers a practical entry point to studying recursive self-improvement (RSI): a propose–evaluate–revise loop can improve this software with fixed model weights (Lee et al., 2026b). This avoids parameter updates and their training infrastructure, although proposal and evaluation costs remain (Zhang et al., 2026c).

The form of this adaptation depends on the task. Surveys of workflow optimization and task-oriented harness design explain how computational structure and model–harness configurations respond to task demands (Yue et al., 2026; Guo et al., 2026b). In harness evolution, these demands shape editable artifacts, interfaces, and feedback. Meta-Harness applies its outer loop to classification, mathematics, and coding using domain-specific instructions and evaluation (Lee et al., 2026b); SkillWeaver evolves reusable browser functions (Zheng et al., 2025a).

We study *harness evolution*: reusable changes to code, configuration, skills, or memory procedures, including training coupled with harness revision. RSI further asks whether a changed mechanism

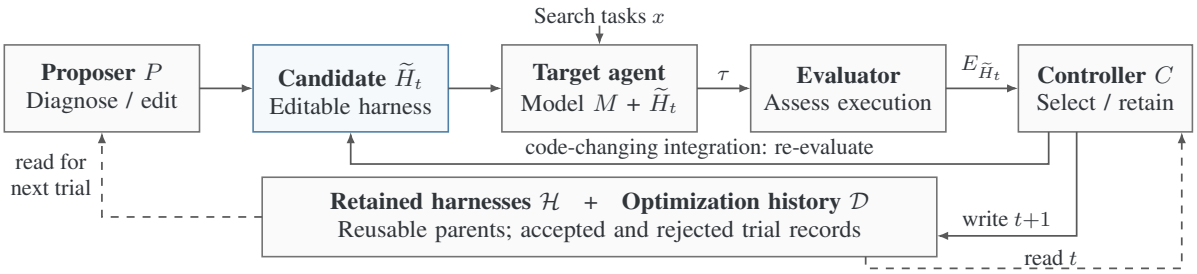


Figure 1: One harness-improvement trial. Solid arrows carry artifacts or updates; dashed arrows read retained state. Task-agent memory inside H differs from optimization history $\mathcal{D}$. Code-changing integration requires re-evaluation. Recursive scope is assessed separately in Section 3.

participates in producing subsequent improvements (Liu et al., 2026d; Wu et al., 2026a). This connects agent engineering to the potential acceleration of automated AI R&D (Chan et al., 2026); task-performance gains alone do not establish this recursive benefit.

Building on this literature, we ask: *what has harness evolution actually produced, and under what task conditions do these changes help?* Our unit of comparison is a reusable harness change within a task subcategory: what the seed could do, what changed, and what the reported evidence establishes. This connects improvement procedures and recursive scope to the concrete functionality they produce.

Our contributions are threefold:

- **A source-linked record of what evolved.** We assemble 95 category–source records across four domains, drawing on 62 research papers and one benchmark resource (Figure 3). Each record connects the benchmark and seed harness to concrete modifications, their reported status, and observed outcomes or evidence limits (Appendices C–F). Readers can trace an evolved capability back to its starting conditions and source evidence.

- **Findings that guide what to evolve and reuse.** Comparing concrete revisions, we identify execution failures repairable outside model reasoning, cases where added tools or checks fail to help, and conditions governing recovery reuse. These findings direct future experiments toward specific interface failures, decision-relevant evidence, and transfer between compatible failure states (Sections 4–5).

- **An account of where recursive reuse occurs.** We compare improvement workflows and identify configurations that reuse an edited agent, improvement skill, or search policy to generate later revisions. Applying the existing five-level scheme (Liu et al., 2026d), we distinguish these configurations from fixed-improver search and assess whether their experiments also establish benefits to subsequent optimization (Sections 2–3).

The collection covers work available by October 1, 2026. Appendix A.1 documents search and selection criteria.

2 Harness Evolution: Workflow and Design Choices

This section asks how execution experience becomes a reusable change to an agent's harness. We define the target and improvement loop, then compare candidate search, memory use, edit guidance, and coupling with training. These are complementary design dimensions, not successive stages or mutually exclusive method classes. The framework covers both fixed-procedure optimization and recursive variants: Section 3 assesses recursive scope, while Section 4 examines the functionality produced. Appendix A supplies supporting comparisons and revision histories.

2.1 Harness Improvement Workflow

Figure 1 separates five components; the computational roles can share a model or implementation. For fixed base model M, target-agent harness H, and task $x \sim \mathcal{X}$, execution produces trajectory $\tau \sim p_M(H, x)$ with reward $r(\tau, x)$. Following Lee et al. (2026b), optimization seeks expected performance across tasks:

$$H^* = \arg\max_H \ \mathbb{E}_{x\sim\mathcal{X},\, \tau\sim p_M(H,x)}[r(\tau, x)]. \tag{1}$$

A proposer constructs candidate $\widetilde{H}_t$; evaluation produces record $E_{\widetilde{H}_t}$; a controller updates retained harnesses $\mathcal{H}_t$ and optimization history $\mathcal{D}_t$. Search uses a finite task set $\mathcal{X}_{\text{search}}$; its scores estimate rather than establish generalization. Appendix A.2 expands the loop; Section 2.5 relaxes the fixed-model setting.

Target agent. The task-executing system is M operating through the harness being optimized. Meta-Harness searches separately for classification, mathematics, and coding, whereas Multi-Task Self-Harness evolves one shared harness across benchmarks (Lee et al., 2026b; Xu, 2026). This choice defines the intended scope of reuse; shared optimization still requires held-out transfer evaluation.

Target agent's harness and editable components. H comprises reusable instructions, interfaces, memory mechanisms, and execution control. Evolution edits designated files or modules within this larger system; it need not rewrite the entire project. AHE separates system prompts, tool descriptions, tool implementations, middleware, skills, subagents, and memory, while fixing verification and logging. Git supports comparison and rollback. Meta-Harness edits a bounded Python program; Multi-Task Self-Harness permits workspace restructuring around a fixed kernel (Lin et al., 2026; Lee et al., 2026b; Xu, 2026). The edit boundary determines which failures proposals can address.

Proposer. The proposer turns prior code and execution evidence into edits within the allowed scope. AHE supplies structured diagnostic reports, whereas Meta-Harness allows active inspection of code, scores, and raw traces through filesystem tools (Lin et al., 2026; Lee et al., 2026b). This contrast concerns the evidence accessible to the editing role; Section 2.4 compares how evidence directs a modification.

Evaluator. Evaluation may use task scores, preferences, or proxies: Meta-Harness, RHO, and TTHE illustrate these alternatives (Lee et al., 2026b; Pan et al., 2026; Nie et al., 2026b). HarnessBank additionally checks whether a changed mechanism activates (Luo et al., 2026e). Activation, proxy improvement, and improved task performance support different conclusions about an edit. Reference-free feedback can guide search when labels are unavailable, but the meaning of an accepted change remains tied to the signal used.

Update controller and history. The controller selects parents, accepts or rejects candidates, and updates retained versions and trial records. Retaining executable parents and retaining evidence are separate decisions: SICA and DGM differ in parent selection, while RRSI retains rejected edits as negative evidence (Robeyns et al., 2025; Zhang et al., 2026c; Xia et al., 2026). A rejected candidate can therefore inform later proposals without remaining eligible for expansion.

2.2 Search Strategies

We compare which harness versions can become parents and how exploration proceeds across their lineages.

Incumbent-centered refinement. SICA selects the highest-utility archived agent; RRSI constrains edits to an incumbent (Robeyns et al., 2025; Xia et al., 2026). AIDE^2 separates levels: its outer loop edits the best research agent, while that agent's inner solution search can change (Srikanth et al., 2026). An archive or an internal tree search does not itself imply exploration of multiple harness parents.

Tree and archive exploration. AFlow retains workflows in a search tree, and DGM can expand alternative archived agents (Zhang et al., 2025b, 2026c). HGM uses descendant outcomes to estimate an ancestry's future value; AgentEvo preserves accuracy–cost trade-offs (Wang et al., 2026b; Xiao et al., 2026). Their common question is why to continue beyond the current best: promising descendants and multiple objectives justify different retention rules. Appendix A.3 compares further search objects and diversity mechanisms.

Parallel lineages. TTHE retains separate branches while sharing execution traces, then selects among their final candidates (Nie et al., 2026b). This separates evidence sharing from parent sharing. Search determines where evolution can continue; memory determines which prior experience can inform that continuation.

2.3 Memory Management

Memory differs in who uses experience and whether updates change its contents or the procedures managing it.

History read by the proposer. Meta-Harness makes trial code, scores, and traces queryable; MetaSkill-Evolve additionally modifies procedures for retrieval and proposal generation (Lee et al., 2026b; Wang et al., 2026f). Accumulating records and changing how the improver uses them are different interventions.

Experience used during deployment. MemoHarness's controller retrieves cases and patterns from a frozen bank to configure the target agent for each new task. During search, the growing bank instead guides harness proposals (Huang et al., 2026b). The consumer and phase therefore matter: deployment reuses learned experience without new feedback or search. Related task-memory methods are compared in Appendix A.3.

Evolving memory procedures. MemEvolve searches encoding, storage, retrieval, and management architectures; MemCodex rewrites executable construction, indexing, retrieval, and routing programs (Zhang et al., 2026a; Wang and Liu, 2026). The updated object is how the target agent receives experience, rather than only the experiences stored.

2.4 Guiding Evolution Directions

Given a parent harness, guidance identifies which deficiency to address and what evidence should justify the edit.

Diagnosing available traces. AHE links structured reports to predicted edit effects; RHO combines individual and cross-trajectory analysis (Lin et al., 2026; Pan et al., 2026). Both turn observed failures into modification hypotheses beyond an aggregate score.

Exposing the next weakness. ActiveSaddler maintains a dynamic failure-pattern registry to revisit known weaknesses or explore new scenarios (Park et al., 2026). This representative case changes what evidence subsequent executions collect, complementing diagnosis of existing traces. Revisiting known failures tests whether a weakness persists, whereas new scenarios can expose blind spots; the registry supplies a concrete way to balance those directions.

Preserving successful behavior. HarnessLens retains successful strategies and selects checks for recovery and regression; ActiveSaddler can revisit earlier successes and rediscover regressions (Xu et al., 2026b; Park et al., 2026). Edit-specific verification and continued exploration provide complementary constraints on direction.

2.5 Workflow Extensions

Coupling with model training. Co-Harness and WHALE use trajectories from improved harnesses to train the target model, which changes the target of later harness search (Chen et al., 2026e; Kim et al., 2026a). Harness-R1 instead applies supervised training followed by GRPO to the editor, using fresh candidate executions for reward while holding the target model fixed (Shao et al., 2026). These loops update different components. Together, the design choices above describe how revisions are produced; recursive reuse and the resulting task capabilities require the separate analyses in Sections 3 and 4.

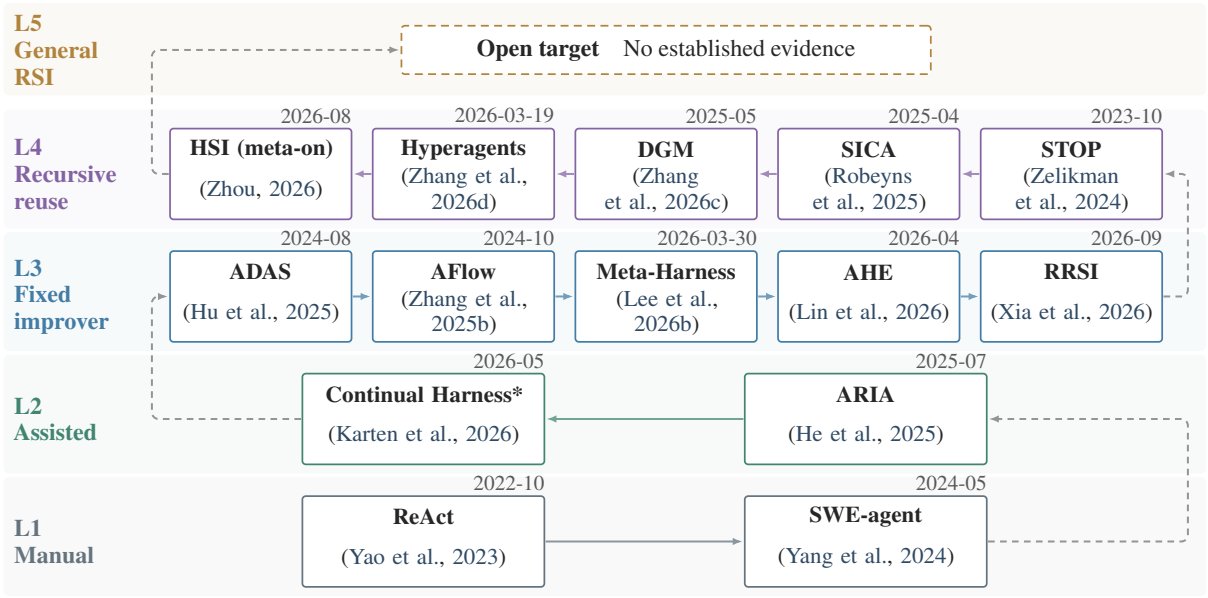


Figure 2: Selected configurations grouped by the five levels of Liu et al. (2026d). Within each level, arrows follow first arXiv release; cross-level links indicate reading order, not chronology or technical inheritance. Dates may differ from citation years; spacing is schematic. *Continual Harness denotes its human-guided GPP configuration. Higher recursive scope need not appear later: early L4 examples coexist with recent L3 methods. L5 remains an open target.

3 Five Levels of Harness Self-Improvement

Following Liu et al. (2026d), we classify *reported configurations* by update autonomy, recursive scope, and generality (Figure 2). Levels do not measure edit breadth or task performance. Appendix B details recursive reuse (Figure 5).

3.1 Capability Levels

L1: Manual improvement. Humans revise the harness, as in the engineered ReAct procedure and SWE-agent interface (Yao et al., 2023; Yang et al., 2024). Autonomous execution does not imply autonomous harness revision.

L2: Assisted improvement. Agents contribute experience, but substantive updates require human decisions. Continual Harness contrasts human-guided revision with an automated refiner (Karten et al., 2026); the configurations therefore have different update autonomy. Expert corrections in ARIA and Growing with Your Embodied Agent similarly supply guidance for persistent updates (He et al., 2025; Meng et al., 2025).

L3: Programmatic self-improvement. The loop runs autonomously under a prescribed improvement procedure. Meta-Harness, AHE, and RRSI differ in search, diagnosis, and acceptance constraints (Lee et al., 2026b; Lin et al., 2026; Xia et al., 2026). Transferable task gains remain possible without revising the mechanism that produces updates.

L4: Bounded recursive self-improvement. A modified component participates in producing subsequent improvements. SICA and DGM reuse modified coding agents; Hyperagents exposes both task and meta agents to editing (Robeyns et al., 2025; Zhang et al., 2026c,d). Recursion remains bounded by fixed components such as evaluation or parent selection. Reuse establishes a mechanism, not automatically an efficiency advantage.

L5: General recursive self-improvement. The improvement mechanism remains effective across broad and changing domains. Hyperagents reports transferred meta-level improvements and continued evolution (Zhang et al., 2026d); these finite experiments support bounded generality. Shared-harness or component

(a) Evidence coverage

62 papers + 1 benchmark resource
95 records; 20 task subcategories

Domain	Subcat.	Records
Coding	4	36
Reasoning and information	7	33
Interactive and embodied	4	17
Visual tasks	5	9

(b) What a record makes traceable

Organized by **task subcategory**, then **source study**

Source setting Benchmark; seed → **Reusable change** Failure and edit → **Result and scope** Outcome and limits

Source locators and artifact status accompany the record.
A record is neither an independent discovery nor an isolated effect.

(c) From reported observations to comparative findings

Execution boundaries

ModularRSI (Wu et al., 2026b)
Multiline payload: shell misrouting → editing-tool dispatch.

SkillWeaver (Zheng et al., 2025a)
Ambiguous textbox lookup → dialog-scoped selector.

Test the execution interface before adding reasoning stages.

Limits of added complexity

Prover interface (Viennot et al., 2026)
Used search: no accuracy gain.

AHE (Lin et al., 2026)
Overlapping closure checks → redundant rechecking.

An activated tool or check need not improve a decision.

Conditions for recovery reuse

Zetta (Ding et al., 2026)
Whole-stack transfer, three tasks: 64% → 84%; no further evolution.

VideoEvolve (Luo et al., 2026a)
Extra checks on the same frames: a light-switch regression persists.

Test reuse under compatible failure states and actions.

Figure 3: The domain evidence set and its analytical use. Counts cover Appendices C–F, with sources deduplicated across categories; records include reporting limits. Selected observations illustrate the findings in Section 4.5. ModularRSI is a reported historical revision; Zetta's gain concerns the complete stack, and VideoEvolve's regression is one case. Panel (c) arrows are not isolated causal estimates.

transfer in Multi-Task Self-Harness and ModularRSI addresses related but distinct questions (Xu, 2026; Wu et al., 2026b). We do not treat these results as an established L5 system.

3.2 Scope and Evidence of Recursive Mechanisms

What changes. Recursion may affect the improver itself, as in STOP; improvement skills, as in MetaSkill-Evolve; or evolver policies, as in HSI (Zelikman et al., 2024; Wang et al., 2026f; Zhou, 2026). The Last Harness You'll Ever Build proposes an evaluation-containing blueprint (Seong et al., 2026). Dream-RSI refines and redeploys an exploration policy, while HGM's prescribed clade-based selection can make different decisions as descendant evidence accumulates (Zheng et al., 2026; Wang et al., 2026b). The former changes the policy; the latter does not by itself establish that the selection rule evolved. These scopes overlap, but evolving task skills, selecting new tasks, or adapting decisions under a fixed rule does not alone establish that the improvement mechanism changed.

What the evidence establishes. AIDE^2 illustrates the distinction between improving an agent and promoting it to the improving role. Its main outer loop edits an inner research agent; a separate ignition test uses an evolved agent as the outer improver. Three seeds per arm do not establish an efficiency advantage (Srikanth et al., 2026, Section 3.6). Accordingly, we distinguish a reusable harness change, execution of a changed improvement mechanism, and demonstrated benefit to subsequent improvement. The comparison should name the changed artifact, identify its later invocation, and state what remained fixed. Meta-level ablations or matched-budget comparisons then address the stronger question of benefit; task-score gains alone cannot answer it. Section 4 examines the concrete artifacts underlying task-level gains.

4 Task Domains, Harness Modifications, and Benchmarks

We organize 95 category–source records from 62 research papers and one benchmark resource into 20 subcategories (Figure 3). Records connect seeds, revisions, and outcomes or reporting limits. This enables

comparison of which failures edits address and whether benefit or reuse was tested. Appendix A.1 defines the counts; Appendices C–F supply records. Figure 10 compares concrete changes across domains, grouped by task objective.

4.1 Coding

Program generation, repository repair, terminal tasks, and ML engineering manipulate executable artifacts through different interfaces (Appendix C; Figure 6). Three modifications recur across these settings.

Routing edits to the intended tool. ModularRSI's dispatcher rejects a multiline `write_file` call and sends its Go code to the shell. The reported revision dispatches by editing-command name and preserves the remaining payload (Wu et al., 2026b). SICA reports editor-input normalization and editing-task routing; SIA's ML experiment driver instead returns structured parameter–score pairs (Robeyns et al., 2025; Hebbar et al., 2026). ModularRSI's three successful runs per version average 36 versus 25 interaction steps, exposing avoidable execution overhead despite identical success. Concurrent edits preclude dispatcher-only attribution; release differences are recorded in Appendix C.

Reading locally and bounding returned output. DGM adds line-range views and unique-string replacement; SICA reduces diff context and limits AST analysis to relevant regions (Zhang et al., 2026c; Robeyns et al., 2025). Multi-task Self-Harness retains the beginning and end of long outputs, further shortens older returns, and suggests narrower commands or saving output to files. Its complete evolved version solves seven of twelve SWE-Bench Pro tasks where the seed exceeds the input limit (Xu, 2026). This is whole-version recovery, not a truncation ablation. AIDE^2 discovers bounded experiment summaries alongside strategy-arm selection and periodic forking (Srikanth et al., 2026). These ML discoveries complement the coding cases: useful context preserves the unit needed for the next decision, whether a code region, diagnostic output, or experiment summary.

Giving distinct failures distinct recovery paths. DGM and TTHE retry empty patches; AFlow routes failed program tests to repair; Self-Harness's GLM instructions request dependency installation and retesting before further source debugging (Zhang et al., 2026c; Nie et al., 2026b; Zhang et al., 2025b, 2026b). In the paired Astropy case, the promoted harness completes executable verification where the seed stops at syntax checks. AHE's shell guard instead blocks cleanup that would destroy verified outputs (Lin et al., 2026). The same failed delivery thus calls for different interventions: recreate an empty patch, repair code, restore the test environment, or preserve an already validated output.

4.2 Reasoning and Information Processing

Mathematics, classification, and QA differ in what makes evidence decisive (Appendix D; Figure 7).

Resolving interpretation before computation. ADAS's discovered arithmetic agent revises a structured problem representation before solving. HarnessEvolve's OfficeQA skill instead specifies operands, denominators, and corpus checks before financial calculations (Hu et al., 2025; Jiang et al., 2026b). Both check interpretation before solving, using call ordering or persistent instructions. AFlow's arithmetic workflow checks a selected answer computationally, moving verification after candidate generation (Zhang et al., 2025b). Checking targets different stages; these artifact observations do not isolate causal gains.

Retrieving evidence that can change a decision. Meta-Harness's draft-verification classifier retrieves same-label confirmers and competing-label challengers; its selected single-call alternative combines label coverage with contrasting examples. Task-CoEvolve fuses word- and character-based rankings while discounting repeated labels (Lee et al., 2026b; Miyai et al., 2026). These are shared artifacts across legal and symptom tasks, not independent discoveries per dataset. MCE additionally routes legal-act warnings or symptom discriminators according to the input (Ye et al., 2026a). EvolveMem activates name-stripped queries and multi-hop decomposition from disabled configuration capabilities; removing either during repeated evolution lowers LoCoMo F1 (Liu et al., 2026b). This supports their contribution within that procedure, not one fixed-checkpoint edit. The shared target is the unresolved distinction between answers: retrieval must expose discriminating evidence, not merely similar or plentiful examples.

Turning proof failures into usable next steps. An evolved Lean lineage adds bounded repairs and checks lemma names and signatures. A richer helper-lemma branch suffers context and assembly failures (Li et al., 2026b). Evolved Rocq interfaces instead add persistent proof states, rollback, and local candidate trials to whole-file compilation; a heavily used lemma-search proposal is rejected without accuracy gains (Viennot et al., 2026). These cases favor feedback that supports a concrete next action, while showing that more tools or decomposition need not improve completed proofs. The reusable repair procedure or interface, rather than an individual proof, is the evolved artifact.

4.3 Interactive and Embodied Tasks

Browser, application, game, and robot tasks change the state that subsequent actions encounter (Appendix E; Figure 8).

Binding actions to interfaces and prerequisites. SkillWeaver repairs an ambiguous textbox lookup by restricting it to the deletion-confirmation dialog, then retains the API patch. LiteEvo's AppWorld harness instead adds login guidance and a reminder after repeated authorization errors (Zheng et al., 2025a; Choi et al., 2026). The former changes executable selection; the latter supplies missing prerequisites at a failure state. LiteEvo's prompt-plus-processor result exceeds either component alone, but this bundle comparison does not isolate its login reminder. These cases locate the failure before action execution: identify the intended object and satisfy its prerequisites before expanding the action sequence.

Moving rules and calculations into decision support. AutoHarness synthesizes game-specific legality code within a supplied action/retry loop. DemoEvolve's Liar's Dice artifacts compute claim probabilities and legal raises; a selected variant overrides continued raising when calling has sufficiently higher expected value (Lou et al., 2026; Che et al., 2026). Legality checks constrain admissible actions, while probability calculations compare their value. The evolved computations must be distinguished from the inherited verification loop and results of other configurations.

Grounding recovery in physical state. Zetta's placement lineage adds bounded regrasp, remembered failed geometries, and support checks before release. Its cumulative stack transfers unchanged to three related tasks, raising mean success from 64% to 84% without isolating each branch (Ding et al., 2026). HarnessPAI instead corrects a basket target using upper-rim points while retaining the placement controller (Wang et al., 2026c). One adds recovery actions; the other repairs a state estimate that makes an existing action fail. Both require interpreting physical observations, beyond detecting a deterministic interface error.

4.4 Visual Understanding and Generation

Visual harnesses preserve which observations support a claim and govern reusable inspection or production procedures (Appendix F).

Keeping claims attached to observations. Video-RSI extends stored observations with frame-linked findings, uncertainty, and entity/event relations. Graph queries identify gaps without processing new frames (Luo et al., 2026b). The full evolved harness improves MLVU accuracy from 62.4% to 72.9% while using fewer frames; neither isolated graph gains nor total-compute savings follow. AutoRef similarly binds requested subjects and scene elements to input references, making generation failures reference-specific (Oshima et al., 2026).

Rechecking without acquiring more observations. VideoEvolve uses the same 64 frames but adds checks for repeated events and interval boundaries. Conditional extra predictions are combined by median endpoints (Luo et al., 2026a). One kettle case improves; a light-switch regression persists. Unlike Video-RSI's selective additional observation, this spends more reasoning on unchanged evidence: repeated checking cannot guarantee recovery from a shared misinterpretation.

Selecting and refining concrete compositions. AutoRef generates subject-led and background-led drafts, checks hard failures, and requires a tied challenger to win in both presentation orders. Reference-specific complaints guide another draft (Oshima et al., 2026). At the same three-image budget, replacing the combined failure-aware selector lowers the held-out average from 7.37 to 6.93; this does not isolate

order swapping. COMFYCLAW and VideoWeaver instead retain composition recipes or production skills for later tasks (Li et al., 2026d; Wei et al., 2026a). Here the reusable gain concerns how later drafts are produced and compared; the revised image itself is a task output.

4.5 Cross-Domain Evaluation and Synthesis

Seeds, edit scope, feedback, and search cost condition comparisons (Ursekar et al., 2026; Huang et al., 2026a); Appendix G details evaluation contracts. We derive three inferences from the preceding cases.

Failures can originate outside the model's reasoning. ModularRSI repairs dispatch of an existing file-edit command; SkillWeaver disambiguates an existing browser action; HarnessPAI corrects a placement target while retaining its controller (Wu et al., 2026b; Zheng et al., 2025a; Wang et al., 2026c). Across these cases, evolution changes how an intended operation reaches its object. This motivates checking payload handling, object identity, and state estimates before adding reasoning stages. Task gains need not indicate a new reasoning capability.

Added complexity need not improve performance. Local code reads, contrasting-label retrieval, and frame-linked observations make evidence usable for different decisions (Zhang et al., 2026c; Lee et al., 2026b; Luo et al., 2026b). Conversely, a heavily used lemma-search proposal is rejected without accuracy gains, and AHE's overlapping checks consume turns redundantly (Viennot et al., 2026; Lin et al., 2026). These cases motivate testing whether each addition improves a decision; activation alone is insufficient. They establish no universal preference for smaller harnesses.

Recovery rules carry conditions for reuse. An empty-patch check can trigger a coding retry; physical recovery needs a grasp or support state; visual rechecking can repeat a mistaken interpretation (Zhang et al., 2026c; Ding et al., 2026; Luo et al., 2026a). Thus, a shared instruction to "retry" hides different detection and action requirements. Zetta demonstrates unchanged reuse across related placement tasks. Cross-domain portability remains untested here: analogies motivate experiments matching failure conditions, observable state, and available actions.

5 Challenges and Future Directions

Three priorities follow from the domain comparisons. Appendix H expands their evidence and separates reliability, transfer, and attribution checks (Figure 12).

Measure what transfers. Transferred components, shared harnesses, and transferred improvers answer different questions (Wu et al., 2026b; Xu, 2026; Zhang et al., 2026d). Evaluations should separate unchanged reuse from further adaptation, with held-out feedback and comparable total cost. Acceptance gates address selection uncertainty (Zhang et al., 2026b; Xia et al., 2026); they do not replace independent final evaluation or accounting for proposal and validation costs.

Attribute gains before expanding search. Mechanism activation and performance benefit are distinct (Luo et al., 2026e). Retained versions, traces, and selective rollback can identify which edits matter and which interact, informing subsequent search. Co-Harness and WHALE further couple model and harness updates (Chen et al., 2026e; Kim et al., 2026a); testing old and updated models under old and updated harnesses can separate their contributions.

Preserve constraints across generations. A local guard, such as AHE's protection against destructive cleanup, does not establish lineage-wide safety (Lin et al., 2026). Revalidation, recovery to earlier versions, and protection of evaluation mechanisms become especially important when the improver itself changes.

Useful revisions address identifiable execution failures and decision needs under specific task conditions. Stronger RSI claims require evidence of improved subsequent optimization beyond better task execution.

Limitations

This survey uses a literature cutoff of October 1, 2026. Targeted searches and reference following provide broad coverage but do not establish exhaustive retrieval; terminology changes rapidly, and publication and artifact-release practices affect which studies can be inspected. Many included works are preprints. Detailed examples favor studies reporting concrete modifications, so the visibility of a mechanism in our synthesis should not be interpreted as its prevalence or superiority. We compare reported evidence rather than independently reproduce all experiments. Released code may differ from evaluated versions, and missing final artifacts or ablations limit causal interpretation. The domain groups organize task objectives and overlap in tools and capabilities. The five-level framework is an analytical lens applied to reported configurations, not a standardized benchmark or proof of general recursive self-improvement. Cross-domain analogies identify design questions; they do not establish portability without transfer evaluation.

Use of AI-Assisted Tools

OpenAI Codex was used to assist with literature organization, source checking, and manuscript and LaTeX editing. AI assistance does not constitute independent experimental reproduction or replace human scholarly judgment. Responsibility for the final content, source interpretation, and submission rests with the human authors.

Funding

This research received no external funding.

Conflicts of Interest

The authors declare no conflicts of interest.

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, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. In *International Conference on Learning Representations*, volume 2026, pages 8479–8565. Case-analysis source version: arXiv:2507.19457v2.

Jiamu Bai, Lizhu Zhang, Xin Yu, Yanhong Wu, Zellux Wang, Serena Li, Weiwei Li, Zhuokai Zhao, Lingzhou Xue, Kiwan Maeng, Xiangjun Fan, and Bo Peng. 2026. Verify Less, Evolve More: Training Idea-Level Critics for Verification-Efficient ML Evolving Agents. *Preprint*, arXiv:2610.08993. ArXiv preprint, version 1; first submitted 6 October 2026; post-cutoff contextual reference.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. 2026. Remember Me, Refine Me: A Dynamic Procedural Memory Framework for Experience-Driven Agent Evolution. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 16803–16822. Association for Computational Linguistics. Survey source: arXiv:2512.10696v1.

Alan Chan, Christoph Winter, Andrew Barto, Jakub Pachocki, Geoffrey Hinton, Eric Horvitz, Yoshua Bengio, Dawn Song, Jack Clark, Hilary Greaves, Anton Korinek, Samuel Hammond, Thore Graepel, Ben Bariach, Philip H. S. Torr, Sheila A. McIlraith, Jeff Clune, Sam Manning, Girish Sastry, and 3 others. 2026. What if automating AI R&D triggers an intelligence explosion? *Preprint*, arXiv:2609.36054. ArXiv preprint, version 1.

Lirong Che, Yuzhe yang, Peiwen lin, Xu Cao, Chuang wang, Xueqian wang, and Jian su. 2026. Demoevolve: Demonstration-guided harness evolution under sparse feedback. *Preprint*, arXiv:2605.24539.

Jianlyu Chen, Yuyang Hu, Hongjin Qian, Jiawei Liu, Wenqing Wei, Xiaolong Chen, Defu Lian, Zhicheng Dou, Chaozhuo Li, Qiwei Ye, and Zheng Liu. 2026a. Repo-To-Skill: Distilling GitHub Repositories Into AI4AI Skills. *Preprint*, arXiv:2609.02749. ArXiv preprint, version 1.

Qian Chen, Chengyuan Liu, and Xin Yu. 2026b. When Should Service Agents Reconsider? Difficulty-Routed Control in Customer-Service Operations. *Preprint*, arXiv:2607.01426. ArXiv preprint, version 1.

Silin Chen, Han Li, Xiaodong Gu, Yuling Shi, and Haibing Guan. 2026c. SkillForge: Self-Distilling Agents for Project-Specific Issue Resolution. *Preprint*, arXiv:2608.18933. ArXiv preprint, version 1.

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, Yuan Xie, Heng Qu, Kun Shao, and Jian Luan. 2026d. HarnessX: A Composable, Adaptive, and Evolvable Agent Harness Foundry. *Preprint*, arXiv:2606.14249.

Zhengyu Chen, Teng Xiao, Huaisheng Zhu, Yige Yuan, Luan Zhang, and Jingang Wang. 2026e. Co-Harness: Co-Evolving Harnesses and Model for Agents. In *COLM 2026 The 2nd Workshop on Lifelong Agents: Learning, Aligning, and Evolving*. Survey source: arXiv:2607.22688v1.

Euntae Choi, Sumin Song, and Sungjoo Yoo. 2026. LiteEvo: Automated, Cost-Efficient Harness Evolution for Generalization to Unseen Tasks. *Preprint*, arXiv:2609.33146. ArXiv preprint, version 1.

Xin Ding, Liang Mi, Mingzhe Huang, Zixuan Wang, Chao Zhang, Zixu Hao, Fu Chen, Xiangyu Li, Yikai Zheng, Yaoyu Guo, Weijun Wang, Kun Li, Hao Wu, Yunxin Liu, and Ting Cao. 2026. Zetta ζ: An Efficient Closed-Loop Embodied Harness for Self-Evolving Physical Intelligence. *Preprint*, arXiv:2608.16590. ArXiv preprint, version 1.

Haoyu Dong, Yuhang Zhou, Zihao Lin, Yifan Wu, Bo Peng, Mingyi Wang, Xiangjun Fan, Lizhu Zhang, and Zhuokai Zhao. 2026. Mixture of Self-Improving Branches For Agent Harness Optimization. *Preprint*, arXiv:2609.37834.

Tianyu Fan and Chao Huang. 2026. HELIX: Model-Harness Co-evolution for Recursive Self-Improvement. *Preprint*, arXiv:2608.13951.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2024. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 13481–13544. PMLR.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, and 8 others. 2026. A Survey of Self-Evolving Agents: What, When, How, and Where to Evolve on the Path to Artificial Super Intelligence. *Transactions on Machine Learning Research*. Survey source: arXiv:2507.21046v4.

Chang Gong, Jingping Bi, Di Yao, Xinjian Liang, Chao Xiang, and Ruijie Guo. 2026. Evolutionary Safety of Recursive Self-Improving AI: Taxonomy, Risk Discovery, and Evaluation. *Preprint*, arXiv:2609.31186.

Hanghui Guo, Weijie Shi, Zhangze Chen, Shengxiang Xu, Yishu Wang, Yimei Zhang, Wangze Ni, Jia Zhu, and Shimin Di. 2026a. DREvo: Distilling Recalibrated Historical Experience for Harness Self-Evolution. *Preprint*, arXiv:2607.26722. Survey source: arXiv:2607.26722v1.

Jianyuan Guo, Zhiwei Hao, Chengcheng Wang, Cheng Fan, Tingzhang Luo, Hongguang Li, Ying Gao, Hefei Mei, Jiankun Peng, Rongjian Xu, Minjing Dong, Han Wu, Mengyu Zheng, Kai Han, Shiqi Wang, Chang Xu, and Yunhe Wang. 2026b. From Question Answering to Task Completion: A Survey on Agent System and Harness Design. *Preprint*, arXiv:2606.20683. ArXiv preprint, version 1.

Yufei He, Ruoyu Li, Alex Chen, Yue Liu, Yulin Chen, Yuan Sui, Cheng Chen, Yi Zhu, Luca Luo, Frank Yang, and Bryan Hooi. 2025. Enabling self-improving agents to learn at test time with human-in-the-loop guidance. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1625–1653, Suzhou (China). Association for Computational Linguistics.

Prannay Hebbar, Yogendra Manawat, Samuel Verboomen, Alesia Ivanova, Selvam Palanimalai, Kunal Bhatia, and Vignesh Baskaran. 2026. SIA: Self Improving AI with Harness & Weight Updates. *Preprint*, arXiv:2605.27276.

Shengran Hu, Cong Lu, and Jeff Clune. 2025. Automated Design of Agentic Systems. In *International Conference on Learning Representations*, volume 2025, pages 21344–21377. Case-analysis source version: arXiv:2408.08435v2.

Lisheng Huang, Chen Yang, Hao Zhou, Huatong Song, Zongchao Chen, Ran Le, Yang Song, Wayne Xin Zhao, and Tao Zhang. 2026a. Evo-Bench: Can Language Models Improve Agent Harness? *Preprint*, arXiv:2608.09096.

Yue Huang, Wenjie Wang, Han Bao, Yuchen Ma, Xiaonan Luo, Yi Nian, Haomin Zhuang, Zheyuan Liu, Yue Zhao, and Xiangliang Zhang. 2026b. MemoHarness: Agent Harnesses That Learn from Experience. *Preprint*, arXiv:2607.14159.

Prithwish Jana, Mononito Goswami, Hao Liu, Xinyu Li, Langlin Huang, Zhehui Huang, Zhishen Huang, Patrick Blöbaum, Anoop Deoras, Purak Jain, Nikos Kanakaris, and Sahika Genc. 2026. Milo: Automated harness discovery via orchestrated multi-agent evolution. *Preprint*, arXiv:2609.38349.

Che Jiang, Jincheng Zhong, Yu Fu, Kai Tian, Junlin Yang, Kaikai Zhao, Yuchong Wang, Tianwei Luo, Weizhi Wang, Yuxin Zuo, Guoli Jia, Xingtai Lv, Dianqiao Lei, Sihang Zeng, Yuru Wang, Zhenzhao Yuan, Xinwei Long, Ermo Hua, Can Ren, and 8 others. 2026a. Self-Improving Agents in the Era of Experience: A Survey of Self- to Meta-Evolution. Public manuscript on OpenReview; author-released PDF dated 25 June 2026.

Wen Jiang, Mingmin Chu, Yimeng Tian, Qianxin Zhang, Haofei Yang, Rui Yang, Yang Liu, Tao Lv, and Fangming Li. 2026b. HarnessEvolve: Learning from Reference Trajectories for Reliable Agent Self-Evolution. *Preprint*, arXiv:2609.00829. ArXiv preprint, version 1.

Borui Kang, Jinrui Gu, Junhan Lv, Wenbin Li, Lei Wang, and Yang Gao. 2026. Harness Continual Learning: Continual Adaptation Beyond Model Parameters. *Preprint*, arXiv:2608.19013. Survey source: arXiv:2608.19013v1.

Seth Karten, Joel Zhang, Tersoo Upaa Jr, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. 2026. Continual Harness: Online Adaptation for Self-Improving Foundation Agents. *Preprint*, arXiv:2605.09998.

Haechan Kim, Yoonho Lee, Gisang Lee, Chelsea Finn, and Kangwook Lee. 2026a. WHALE: A Simple Recipe for Joint Harness-Weight Optimization. *Preprint*, arXiv:2609.00196.

Seungyeon Kim, Junhoo Lee, Minkyu Kim, Baekseung Kim, and Nojun Kwak. 2026b. Recursive Harness Distillation across Agents for Robot Manipulation. *Preprint*, arXiv:2609.33378. ArXiv preprint, version 1.

Hyunin Lee, Jinglue Xu, Jeffrey Seely, Donghyun Lee, Matei Zaharia, and Yujin Tang. 2026a. Recursive Harness Self-Improvement. *Preprint*, arXiv:2607.15524.

Yoonho Lee, Roshen Sanjay Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. 2026b. Meta-Harness: End-to-End Optimization of Model Harnesses. In *Third Conference on Language Modeling*. Survey source: arXiv:2603.28052v1.

Yanzhou Li, Yiran Zhang, Xiaoyu Zhang, Xiaoxia Liu, and Yang Liu. 2026a. CODESKILL: Learning Self-Evolving Skills for Coding Agents. *Preprint*, arXiv:2605.25430. ArXiv preprint, version 2.

Yuqing Li, Zeguan Wu, Yu Gan, and Junyu Liu. 2026b. Self-Modifying Lean Proof Agents with Verifier-Grounded Benchmark Coevolution. *Preprint*, arXiv:2607.17352. ArXiv preprint, version 1.

Zhuochun Li, Youngmin Ko, Ali Keramati, Tuhin Kundu, Liang-Chun Tsai, Nicola Ferri, Mirco Milletari, Jiaxiang Liu, Susana Palmaz Lopez Pelaez, Yuepeng Wang, Vadim Smolyakov, Xiang Jiang, Kjartan Olafsson, and Tommy Guy. 2026c. One success isn't reliability: Thinkingbox, a sandbox and benchmark for agents in stateful business workflows. *arXiv preprint arXiv:2608.19741*. Version 4 (October 1, 2026); Table 5 used for reliability figures.

Zongxia Li, Dawei Liu, Fuxiao Liu, Yuhang Zhou, Xiyang Wu, Jingxi Chen, Jing Xie, Xiaomin Wu, and Lichao Sun. 2026d. COMFYCLAW: Self-Evolving Skill Harnesses for Image Generation Workflows. In *ECCV 2026 Workshop on Multimodal Large Language Models for Unified Comprehension and Generation*. Survey source: arXiv:2607.01709v1; Workshop acceptance verified; final proceedings metadata not confirmed.

Susan Liang, Jianmin Wu, and Daxiang Dong. 2026. VidHarness: Evolving Agent Harnesses for Cost-Efficient Long Video Understanding. *Preprint*, arXiv:2609.38413. ArXiv preprint, version 1.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. 2026. Agentic Harness Engineering: Observability-Driven Automatic Evolution of Coding-Agent Harnesses. *Preprint*, arXiv:2604.25850.

Hongjun Liu, Yifei Ming, Shafiq Joty, and Chen Zhao. 2026a. Harnessing LLM Agents with Skill Programs. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*. Accepted for publication; forthcoming; Survey source: arXiv:2605.17734v2.

Jiaqi Liu, Xinyu Ye, Peng Xia, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. 2026b. EvolveMem:Self-Evolving Memory Architecture via AutoResearch for LLM Agents. *Preprint*, arXiv:2605.13941.

Junxiang Liu, Lin Wang, Haiyu Shi, Hongxu Ma, Xiaoyu Yang, Chunjie Chen, Xiaoxiao Xu, Kaiqiao Zhan, Boao Wang, Shuizhou Shi, Tianyun Zhu, Jie Li, and Jiangtong Li. 2026c. CineForge: Self-Improving Agents for Long-Horizon Video Generation. *Preprint*, arXiv:2608.29621. ArXiv preprint, version 1.

Shuaiqi Liu, Zhengkai Lin, Yuxiang Zhang, Yuanyi Ren, Yue Wu, Yongbin Li, Zheng Wang, Zhihang Fu, and Jieping Ye. 2026d. The Path to Recursive Self-Improving Agents: Foundation, Framework, and Future Directions. Review preprint.

Zhuo Liu, Moxin Li, Zhixin Ma, Wentao Shi, Wenjie Wang, and Fuli Feng. 2026e. HARDE: Optimizing Agent Harnesses for Runtime Risk Detection and Execution Control. *Preprint*, arXiv:2609.38291. ArXiv preprint, version 1.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin Murphy. 2026. AutoHarness: improving LLM agents by automatically synthesizing a code harness. In *ICLR 2026 Workshop on Recursive Self-Improvement*. Survey source: arXiv:2603.03329v1.

Bingjun Luo, Yuhuan Fan, Jialin Guo, and Siqi Li. 2026a. VideoEvolve: Evolving Agent Harnesses for Video Temporal Grounding. *Preprint*, arXiv:2610.01766. ArXiv preprint, version 1.

Bingjun Luo, Jialin Guo, and Siqi Li. 2026b. Video-RSI: Recursive Self-Improvement of Video Understanding Agents via Harness Evolution. *Preprint*, arXiv:2609.37950. ArXiv preprint, version 1.

Haochen Luo, Yi Huang, Sichun Luo, Fengyuan Liu, Lei Li, Zefa Hu, Junlan Feng, and Qi Liu. 2026c. Harness-Aware Self-Evolving: Co-Evolving Model Weights, Harness, and Task Solutions. *Preprint*, arXiv:2607.03935. ArXiv preprint, version 1.

Jiabin Luo, Yinan Liu, Chunlei Meng, and Yufei Guo. 2026d. EvoGen-Harness: Learning Where and How to Evolve Image-Generation Harnesses. *Preprint*, arXiv:2610.00383. ArXiv preprint, version 1.

Xiaotian Luo, Dizhan Xue, Fengxingyu Wang, Chuanrui Hu, and Yafeng Deng. 2026e. HarnessBank: Semantic Gene-Bank Search with Gated Verification for Agent-Harness Self-Evolution. *Preprint*, arXiv:2607.13683.

Fanqing Meng, Lingxiao Du, Qiguang Chen, Ziqi Zhao, Haocheng Lu, Mengkang Hu, and Michael Qizhe Shieh. 2026. Rsibench-data: Benchmarking data-centric research for recursive self-improvement. *arXiv preprint arXiv:2607.25886*.

Yuan Meng, Zhenguo Sun, Max Fest, Xiangtong Yao, Xukun Li, Zhenshan Bing, and Alois Knoll. 2025. Robotic Ultra-Long-Horizon Manipulation Skills via Human-guided Lifelong Code Generation. *Preprint*, arXiv:2509.18597. Case-analysis source: arXiv version 2 (September 25, 2025).

Atsuyuki Miyai, Kiyoharu Aizawa, and Toshihiko Yamasaki. 2026. Task-CoEvolve: Efficient Harness Optimization via Adaptive Validation Task Selection. *Preprint*, arXiv:2608.20169. ArXiv preprint, version 2.

Subhojyoti Mukherjee and Md Mehrab Tanjim. 2026. MoMHa: Multi-Objective Optimization of LLM Harnesses over Accuracy, Safety, and Tokens. In *Advances in Neural Information Processing Systems*. Accepted for publication; forthcoming; Survey source: arXiv:2609.30967v1.

Jun Nie, Yonggang Zhang, Qianshu Cai, Yiu-ming Cheung, Xinmei Tian, and Bo Han. 2026a. EvolveNet: Collaborative Harness Evolution for Agent Self-Improvement. *Preprint*, arXiv:2608.04968. ArXiv preprint, version 1.

Jun Nie, Yonggang Zhang, Jun Song, Qianshu Cai, Dahai Yu, Yike Guo, Xinmei Tian, and Bo Han. 2026b. TTHE: Test-Time Harness Evolution. *Preprint*, arXiv:2607.08124.

Yuta Oshima, Ku Onoda, Yusuke Iwasawa, Masahiro Suzuki, Yutaka Matsuo, and Hiroki Furuta. 2026. AutoRef: Harness Optimization for Agentic Multi-Reference Image Generation. *Preprint*, arXiv:2609.35530. ArXiv preprint, version 1.

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. 2026. ReasoningBank: Scaling Agent Self-Evolving with Reasoning Memory. In *The Fourteenth International Conference on Learning Representations*. Survey source: arXiv:2509.25140v2.

Wenbo Pan, Shujie Liu, Chin-Yew Lin, Jingying Zeng, Xianfeng Tang, Xiangyang Zhou, Yan Lu, and Xiaohua Jia. 2026. Evolving Agents in the Dark: Retrospective Harness Optimization via Self-Preference. In *Findings of the Association for Computational Linguistics: EMNLP 2026*. Accepted for publication; forthcoming; Survey source: arXiv:2606.05922v3.

Sungho Park, Wonjoong Kim, Jue Zhang, Wook-Shin Han, Pengfei Gao, Chanyoung Park, Yongqiang Yao, Rao Fu, Elsie Nallipogu, Qingwei Lin, and Victor Rühle. 2026. ActiveSaddler: Automated Curriculum Learning for Agent Harness Optimization. *Preprint*, arXiv:2610.00906. ArXiv preprint, version 1.

Zeyu Ren, Ling Yue, Ran Li, Yishu Wang, Shengxiang Xu, Hanmo Liu, Shaowu Pan, and Shimin Di. 2026. FlowEvo: Self-Evolving Agents through the Co-Evolution of Workflows and Executable Skills. In *Third Conference on Language Modeling (COLM)*. Survey source: arXiv:2607.21596v2.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. 2025. A Self-Improving Coding Agent. In *ICLR 2025 Workshop on Scaling Self-Improving Foundation Models without Human Supervision*. Survey source: arXiv:2504.15228v2.

RSI-Exam contributors. n.d.a. RSI-Exam: Lean 4 Proof-Search LLM Agent Workflow Design. Task specification and public optimization traces; 2026 resource snapshot, publication date not stated.

RSI-Exam contributors. n.d.b. RSI-Exam: LLM Agent Memory Architecture Design. Task specification and public optimization traces; 2026 resource snapshot, publication date not stated.

RSI-Exam contributors. n.d.c. RSI-Exam: Scientific Discovery Agent Harness Design. Task specification; 2026 resource snapshot, publication date not stated.

Scale AI. n.d. RSI Bench: Agent Swarm Optimization. Task specification; 2026 resource snapshot, publication date not stated.

Biswa Sengupta and Jinhua Wang. 2026. HARBOR: Automated Harness Optimization. *Preprint*, arXiv:2604.20938. ArXiv preprint, version 1.

Haebin Seong, Li Yin, Haoran Zhang, and Zhan Shi. 2026. The Last Harness You'll Ever Build. *Preprint*, arXiv:2604.21003. ArXiv preprint, version 3.

Shuai Shao, Kangning Zhang, Qingyao Li, Shijian Wang, Hao Wang, Wenxiang Jiao, Yuan Lu, Yi Guo, Weiwen Liu, and Weinan Zhang. 2026. Harness-R1: Learning to Edit Executable Runtime Harnesses from Agent Failure Trajectories. *Preprint*, arXiv:2608.02276.

Dhruv Srikanth, Bingchen Zhao, Dixing Xu, Yuxiang Wu, and Zhengyao Jiang. 2026. Recursive self-improvement of AI research agents. *Preprint*, arXiv:2609.26457.

Varun Ursekar, Apaar Shanker, Yash Maurya, Shehab Yasser, Vijay S. Kalmath, Veronica Chatrath, and Yuan Xue. 2026. HarnessOpt-Bench: Evaluating LLMs at Harness Optimization. *Preprint*, arXiv:2608.06301.

Naman Vats and Oleg Aleksandrovich Golev. 2026. The Scaffold Effect in Coding Agents: Harness Choice as a Hidden Variable in Coding-Agent Evaluation. In *KDD workshop on Evaluation and Trustworthiness of Agentic AI*. Survey source: arXiv:2607.22585v1.

Jules Viennot, Guillaume Baudart, and Marc Lelarge. 2026. Growing an Agent/Prover Interface: Evolutionary Tool Design for Cost-Efficient Theorem Proving in Rocq and Lean. *Preprint*, arXiv:2609.39544. Survey source: arXiv:2609.39544v1.

Peidong Wang, Zhiming Ma, Ying Chang, Xufang Luo, Yiqun Zhang, Zihan Wang, Xiaocui Yang, Shi Feng, Yuqing Yang, and Dongsheng Li. 2026a. Self-Evolving Embodied Agents via Skill-Harness Evolution. *Preprint*, arXiv:2608.11350. Survey source: arXiv:2608.11350v1.

Wenyi Wang, Piotr Piękos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. 2026b. Huxley-Gödel Machine: Human-Level Coding Agent Development by an Approximation of the Optimal Self-Improving Machine. In *The Fourteenth International Conference on Learning Representations*. Survey source: arXiv:2510.21614v3.

Xiaoqiang Wang and Bang Liu. 2026. MemCodex: Self-Programming Hierarchical Memory for Language Agents. *Preprint*, arXiv:2609.39765. ArXiv preprint, version 1.

Xin Wang, Wenhao Wu, Menghao Zhang, Zhi Wang, Kun Shao, Jian Luan, Yang Li, Qing Li, Shangding Gu, Huichi Zhou, Shuqing Shi, Fei Ni, Shuo Lu, Weicheng Meng, Kang Li, Jin Wu, Kang Zhao, Shangmin Guo, Gen Li, and 4 others. 2026c. HarnessPAI: An Evolving Harness for Physical AI. *Preprint*, arXiv:2609.29166. ArXiv preprint, version 1.

Yihao Wang, Linhan Xia, Rui Liu, Zhaofeng Zhang, Hongyu Wu, Yang Yang, Jinglu He, Yu Guo, and Kai Lei. 2026d. SAGE: A Statistical Acceptance Gate for Self-Evolving Agents. *Preprint*, arXiv:2609.36043. ArXiv preprint, version 1.

Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Pradeep Dasigi, and Teng Xiao. 2026e. Rethinking the Evaluation of Harness Evolution for Agents. In *COLM 2026 The 2nd Workshop on Lifelong Agents: Learning, Aligning, and Evolving*. Version 3 (October 1, 2026); Table 1 used for matched-budget comparisons; Survey source: arXiv:2607.12227v3.

Zefeng Wang, Minxi Yan, Jinhe Bi, Sikuan Yan, Volker Tresp, and Yunpu Ma. 2026f. MetaSkill-Evolve: Recursive Self-Improvement of LLM Agents via Two-Timescale Meta-Skill Evolution. *Preprint*, arXiv:2607.05297.

Jianhui Wei, Yan Zhang, Jie Tan, Hengchuan Zhu, Xiaotian Zhang, Ziyi Chen, Daoan Zhang, Wei Xu, Yeying Jin, and Zuozhu Liu. 2026a. VideoWeaver: Evaluating and Evolving Skills for Agentic Long Video Generation. *Preprint*, arXiv:2606.08091. ArXiv preprint, version 2.

Tianxin Wei, Zhan Shi, Minhua Lin, Bing He, Zewen Liu, Yisi Sang, Yuanchen Bei, Xuying Ning, Jiaru Zou, Ting-Wei Li, Xiao Lin, Yanjun Zhao, Chi Wang, Benoit Dumoulin, Dakuo Wang, Jingrui He, and Hanqing Lu. 2026b. Evo-Harness: Context-to-harness skill compilation for self-evolving agents. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*. Accepted for publication; forthcoming; Survey source: arXiv:2608.15071v1.

Hao Wu, Fan Xu, Fan Zhang, Zhipeng Xu, Yongheng Zhang, Yinghui Li, Yizhou Zhao, Penghao Zhao, Weiran Yao, Zengxiang Li, Yuan Gao, Zhangyang Gao, Hanbin Wang, Ruihan Tao, Wenjie Wang, Qi Li, Junfeng Fang, Xingjun Ma, Qiankun Li, and 13 others. 2026a. Recursive Self-Improvement in AI: A Survey. Preprints.org preprint, version 1; posted 30 September 2026; not peer-reviewed.

Siwei Wu, Jincheng Ren, Yizhi Li, Haau-Sing Li, Chengran Yang, Yuxuan Zhang, Weicheng Gu, Jian Yang, Riza Batista-Navarro, Chuanyi Zhang, Xianglong Liu, Ming Zhou, Bryan Dai, and Chenghua Lin. 2026b. ModularRSI: Modular and Generalizable Recursive Harness Self-Improvement. *Preprint*, arXiv:2609.14857.

Yukai Wu, Yuanjing Yang, Le Zhou, Shaokun Han, Haoyu Wang, Zirui Tang, Xuzhou Zhu, Weihuang Zheng, Maxm Pan, Xuanhe Zhou, and Fan Wu. 2026c. Breaking the Environment Wall: A Unified Framework for Preparing and Evolving Agent-Native Environments. *Preprint*, arXiv:2609.29773. Survey source: arXiv:2609.29773v1.

Peng Xia, Rujun Han, Zifeng Wang, Yanfei Chen, Yufan Zhuang, Yoonho Lee, Chengsong Huang, Han Yu, Zhongying CuiZhu, Yifei Ming, Huaxiu Yao, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. 2026. RRSI: Regularized Recursive Self-Improvement of Agent Harnesses. *Preprint*, arXiv:2609.24972. Survey source: arXiv:2609.24972v2.

Bo Xiao, Haiyang Liu, Yitong Wang, Yanning Hou, Ke Xu, and Xingyi Zhang. 2026. AgentEvo: cost-aware agentic workflow generation via adaptive multi-stage evolution. *Complex & Intelligent Systems*, 12(9):208.

Beibei Xu, Yutong Ye, Chuyun Shen, Yingbo Zhou, Cheng Chen, and Mingsong Chen. 2026a. HyEvo: Self-Evolving Hybrid Agentic Workflows for Efficient Reasoning. *Preprint*, arXiv:2603.19639. ArXiv preprint, version 1.

Jinghan Xu, Yikai Zhang, Aili Chen, Weiyuan Li, Jiaqing Liang, and Deqing Yang. 2026b. Verify Smarter, Evolve Further: Efficient Harness Evolution through Behavior-Aware Verification. *Preprint*, arXiv:2608.27311.

Qiankai Xu. 2026. Self-Evolving Harness on Multiple Tasks with the Agent as Its Own Optimizer. *Preprint*, arXiv:2609.38372. ArXiv preprint, version 1.

Chen Yang, Jiashuo Tian, Ziqi Wang, Xinyin Liu, Meiru Ye, and Junjie Chen. 2026a. Learning Globally Reusable Skills for Coding Agents. *Preprint*, arXiv:2608.06153. ArXiv preprint, version 1.

Jingbo Yang, Kwei-Herng Lai, Xiaowen Wang, Yaar Harari, Evgeniy Gabrilovich, and Shiyu Chang. 2026b. Learning from Research: Toward Lifelong Agent Harness Evolution. *Preprint*, arXiv:2609.40169. ArXiv preprint, version 1.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*, volume 37, pages 50528–50652. Curran Associates, Inc. Case-analysis source version: arXiv:2405.15793v3.

Siqi Yang, Qianlan Yang, Yu-Xiong Wang, Saurabh Pujar, and Martin Hirzel. 2026c. One Recipe, Many Harnesses: What Self-Evolution Encodes Across Languages and Models. In *NeurIPS 2026 Workshop on Towards Test-Time Continual Learning Agents*. Survey source: arXiv:2608.10178v1; Accepted workshop paper; forthcoming.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. 2026d. SkillOpt: Executive Strategy for Self-Evolving Agent Skills. *Preprint*, arXiv:2605.23904. ArXiv preprint, version 2.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*. Source version: arXiv:2210.03629v3.

Haoran Ye, Xuning He, Vincent Arak, Haonan Dong, and Guojie Song. 2026a. Meta Context Engineering via Agentic Skill Evolution. In *Forty-third International Conference on Machine Learning*. Survey source: arXiv:2601.21557v2.

Wenqian Ye, Ziwei Guan, Eric Xie, Bohan Liu, Shivani Modi, Buyun Zhang, Ellie Dingqiao Wen, Henry Kautz, and Aidong Zhang. 2026b. ProofEvolve: Neuro-Symbolic Evolution for Formal Automated Theorem Proving. *Preprint*, arXiv:2608.26334. ArXiv preprint, version 1.

Yongshi Ye, Tian Lan, Feihu Jiang, Muyang Ye, Bin Zhu, Qianghuai Jia, Longyue Wang, Zhao Xu, Weihua Luo, and Xiaodong Shi. 2026c. CHIME: Credit-Aware Hierarchical Memory Evolution for Long-Horizon Agentic Planning. *Preprint*, arXiv:2609.02074. ArXiv preprint, version 1.

Xin Yu, Hanwen Xing, and Lingzhou Xue. 2026a. EXACT: Explicit Attribute-Guided Decoding-Time Personalization. *Preprint*, arXiv:2602.17695. ArXiv preprint, version 1; first submitted 6 February 2026.

Xin Yu, Lizhu Zhang, Jiamu Bai, Yanhong Wu, Zellux Wang, Serena Li, Weiwei Li, Lingzhou Xue, Xiangjun Fan, and Bo Peng. 2026b. MLToolBench: Learning Tool-Augmented Agents for Machine Learning Development. *Preprint*, arXiv:2609.36679. Survey source: arXiv:2609.36679v1.

Ling Yue, Kushal Raj Bhandari, Ching-Yun Ko, Dhaval Patel, Shuxin Lin, Nianjun Zhou, Jianxi Gao, Pin-Yu Chen, and Shaowu Pan. 2026. From Static Templates to Dynamic Runtime Graphs: A Survey of Workflow Optimization for LLM Agents. In *Findings of the Association for Computational Linguistics: EMNLP 2026*. Accepted for publication; forthcoming; Survey source: arXiv:2603.22386v1.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. 2024. Self-Taught Optimizer (STOP): Recursively Self-Improving Code Generation. In *First Conference on Language Modeling*. Survey source: arXiv:2310.02304v3.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. 2025a. Multi-agent Architecture Search via Agentic Supernet. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 75834–75852. PMLR. Case-analysis source version: arXiv:2502.04180v2.

Guibin Zhang, Haotian Ren, Chong Zhan, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. 2026a. MemEvolve: Meta-Evolution of Agent Memory Systems. In *Forty-third International Conference on Machine Learning*. Survey source: arXiv:2512.18746v1.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. 2026b. Self-Harness: Harnesses That Improve Themselves. *Preprint*, arXiv:2606.09498.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. 2026c. Darwin Gödel Machine: Open-Ended Evolution of Self-Improving Agents. In *International Conference on Learning Representations*, volume 2026, pages 104223–104294. Case-analysis source version: arXiv:2505.22954v3.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. 2026d. Hyperagents. *Preprint*, arXiv:2603.19461.

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. 2025b. AFlow: Automating Agentic Workflow Generation. In *International Conference on Learning Representations*, volume 2025, pages 34040–34077. Case-analysis source version: arXiv:2410.10762v4.

Luan Zhang, Ruochen Zhou, Dandan Song, Zhengyu Chen, Yuhang Tian, Jun Yang, Huipeng Ma, Chenhao Li, Guangyuan Feng, Xudong Li, Yizhou Jin, and Yan Xu. 2026e. HarnessCompass: Guiding Automatic Harness Evolution toward Generalizable and Effective Agent Harnesses. *Preprint*, arXiv:2608.01918.

Ruiyi Zhang, Peijia Qin, Qi Cao, Li Zhang, and Pengtao Xie. 2026f. AIBuildAI-2: A Knowledge-Enhanced Agent for Automatically Building AI Models. *Preprint*, arXiv:2605.27873. ArXiv preprint, version 1.

Yifan Zhang, Yutong Dai, Juntao Tan, Luyu Yang, Rishi Mullur, Thai Hoang, Zhiyuan Hu, James Zhu, Phil Mui, Silvio Savarese, Ran Xu, and Zeyuan Chen. 2026g. DarwinX: Evolving Agent Harnesses Through Natural Selection. *Preprint*, arXiv:2608.07545.

Yunbei Zhang, Janet Wang, Saiyue Lyu, Yingqiang Ge, Kaiqu Liang, Zijian Jin, Chandan K Reddy, and Jihun Hamm. 2026h. Safety must survive self-improvement: Why failures persist and how agents recover. *arXiv preprint arXiv:2610.01073*.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. 2025a. SkillWeaver: Web Agents can Self-Improve by Discovering and Honing Skills. *Preprint*, arXiv:2504.07079.

Chengqi Zheng, Jianda Chen, Yueming Lyu, Wen Zheng Terence Ng, Haopeng Zhang, Yew-Soon Ong, Ivor Tsang, and Haiyan Yin. 2025b. MermaidFlow: Redefining Agentic Workflow Generation via Safety-Constrained Evolutionary Programming. In *ICML 2025 Workshop on Multi-Agent Systems in*

the Era of Foundation Models: Opportunities, Challenges and Futures. Non-archival workshop; Survey source: arXiv:2505.22967v1.

Tong Zheng, Xidong Wu, Zheng Zhang, Zhankui He, Chaoyi Zhang, Benjamin Coleman, Ruoqiao Wei, Di Bai, Haolin Liu, Rui Liu, Xue Wang, Yue Zhuan, Wang-Cheng Kang, Renkai Xiang, Heng Huang, Xinwu Cheng, and Yunsong Guo. 2026. Dream-RSI: Recursive Self-Improvement through Evolving Worlds. *Preprint*, arXiv:2609.14858. Survey source: arXiv:2609.14858v1.

Tailin Zhou. 2026. Hierarchical Self-Improvement: A Framework for Task-Specific Evolvable Agent Harnesses. *Preprint*, arXiv:2608.08466.

Xiaofan Zhou and Lu Cheng. 2026. Streamlined Reflective Evolution for Task-Adaptive Self-Refinement Pipelines. *Preprint*, arXiv:2609.32458. ArXiv preprint, version 1.

Guide to the Appendices

The appendices follow the questions developed in the main paper. They provide method comparisons, concrete revision records, and evaluation details in the following order.

- **Workflow (Section 2).** Appendix A gives the collection criteria, formal loop, four design comparisons, and illustrative revision histories.
- **Recursive scope (Section 3).** Appendix B separates update autonomy, the mechanism reused in later improvement, and the strength of the supporting evidence.
- **Four task domains (Sections 4.1–4.4).** Appendices C–F cover coding, reasoning and information processing, interactive and embodied tasks, and visual tasks. Each proceeds from task families and seed harnesses to study-level evolution records.
- **Cross-domain comparison (Section 4.5).** Appendix G compares concrete mechanisms across domains and records evaluation contracts. It supports the synthesis rather than defining a fifth domain.
- **Research priorities (Section 5).** Appendix H follows the main text's three priorities: measure transfer, attribute gains, and preserve constraints across generations.

Within each task subcategory, records identify the setting and seed, reported reusable change, and observed outcome or evidence limit. Repeated evaluation of one artifact is not an independent discovery. Coding records link to the source index in Appendix C.5; other records provide source locators alongside their citations. Overview figures remain with the appendix whose evidence they summarize.

A Supporting Evidence for Harness Workflows

This appendix supports Section 2. Appendix A.1 states the collection and evidence criteria used throughout the survey. Appendix A.2 expands the workflow and maps literature to its components; Appendix A.3 follows the four design themes; Appendix A.4 traces four illustrative updates from reported problems to observed outcomes. Recursive reuse is assessed separately in Appendix B.

A.1 Literature Collection and Evidence Selection

Coverage and retrieval. The collection uses October 1, 2026 as the cutoff for paper availability; later checks recover earlier omissions and verify sources without extending that cutoff. Discovery combines targeted title and topic searches, reference following, and searches by task category and benchmark. Search concepts include harness optimization, recursive self-improvement, workflow search, and skill or memory evolution. Primary sources include arXiv and publisher or proceedings records, author-released manuscripts, and official project repositories. Preprints remain identified as such, and versions of one work are not treated as independent studies. Refereed publications are cited when available; version-specific source notes identify the manuscripts used for the case analysis.

Eligibility and comparison unit. Core studies report automated or assisted revision of a reusable target-agent harness or its improvement mechanism. Static designs, benchmark resources, weight-only adaptation, and related memory or search methods provide context where relevant; they do not independently establish a harness-evolution finding. A task-specific answer, patch, or generated image counts only when the work also changes a procedure reused in subsequent execution. Task-domain cases are grouped by subcategory and study, with benchmark-specific evidence retained within each record; multiple evaluations of one artifact are not independent discoveries.

Size of the domain evidence set. Counts refer specifically to the study entries in Appendices C–F, excluding background references, workflow comparisons, and Appendix G's benchmark inventory. There are 95 category–source records: 36 coding, 33 reasoning/information, 17 interactive/embodied, and nine

visual, across 4, 7, 4, and 5 leaf subcategories respectively. Source identities, checked against arXiv identifiers across publication versions, yield 62 distinct research papers and the RSI-Exam Lean resource. Multiple settings grouped in one category entry count once; a source appearing in different categories contributes multiple records but one source. Records include incomplete artifact reporting, not only demonstrated modifications. These are current collection counts, not retrospective screening totals or a count of independent effects.

Evidence extraction. The case analysis records the seed capability, reported problem, modified artifact, observed behavior or outcome, and evidence location. Detailed examples are selected for an identifiable change and informative contrast, rather than citation count or publication priority. We distinguish supplied mechanisms, proposed or intermediate changes, retained artifacts, and changes whose retention is unclear. Method sections, appendices, and, where checked, public implementations support these distinctions. Unchecked implementation details are not treated as absent from the source. Whole-system gains, component ablations, and individual observations support different strengths of inference; the main text reserves cross-study patterns for substantively comparable evidence. This is an evidence-guided synthesis rather than an exhaustive systematic review or a pooled performance analysis.

A.2 Workflow Components and Formal Loop

Problem formulation. Equation 1 follows the fixed-model objective in Lee et al. (2026b, Section 3): improve expected task reward by revising the reusable harness. In practice, candidate search uses a finite task set $\mathcal{X}_{\text{search}}$, so gains on that set must be distinguished from generalization to the target distribution $\mathcal{X}$. To describe the search for H^*, we use P for the proposer, $\mathcal{H}$ for retained harnesses, $\mathcal{D}$ for the history of code, scores, and traces, and E_H for evaluation records. The time indices and controller C below are survey abstractions.

Improvement loop. A single-candidate round consists of three steps.

Propose a candidate. At round t, the proposer uses the selected parent code, traces, and feedback in optimization history $\mathcal{D}_t$ to construct a candidate harness:

$$\widetilde{H}_t \sim P(\mathcal{D}_t). \tag{2}$$

Evaluate the candidate. The target agent executes tasks from $\mathcal{X}_{\text{search}}$ using $\widetilde{H}_t$. Evaluation records the feedback permitted by the protocol, including task scores, proxies, or execution failures:

$$E_{\widetilde{H}_t} = \text{Evaluate}(\widetilde{H}_t, M, \mathcal{X}_{\text{search}}). \tag{3}$$

Update the retained state. The controller uses this evidence to update the retained harnesses $\mathcal{H}_t$ and the history available to later proposals:

$$\begin{aligned}(\mathcal{H}_{t+1}, \mathcal{D}_{t+1}) \\ = C(\mathcal{H}_t, \mathcal{D}_t, \widetilde{H}_t, E_{\widetilde{H}_t}).\end{aligned} \tag{4}$$

The retained set may contain one incumbent or multiple candidates. Rejected trials can remain in $\mathcal{D}_t$ as evidence for subsequent edits; this optimization history is distinct from memory used by the target agent to solve tasks. Joint model–harness optimization additionally updates M_t (Section 2.5).

Prepared diagnosis and active inspection. AHE's seven edit types separate system prompts, tool descriptions, tool implementations, middleware, skills, subagent configurations, and long-term memory; tracing, verification, run records, and model configuration remain fixed. Git supports comparison and rollback (Lin et al., 2026, Sections 3.1–3.3). RHO combines individual and cross-trajectory diagnosis, whereas Meta-Harness exposes raw trial artifacts for inspection (Pan et al., 2026; Lee et al., 2026b). These choices determine which analysis is supplied and which the proposer performs itself.

Component map. Figure 4 indexes mechanisms by their role in the loop. Its branches overlap: they identify what a method contributes, not mutually exclusive paper classes. The same implementation may perform several roles, and its role can change between search and deployment.

Component	Design dimension	Selected mechanisms and literature
Target agent	**Task coverage**	*Coding agents:* **SICA** (Robeyns et al., 2025), **DGM** (Zhang et al., 2026c). *Separate classification, math, and coding searches:* **Meta-Harness** (Lee et al., 2026b).
	Transfer and coupling	*One harness across tasks:* Multi-Task Self-Harness (Xu, 2026). *Joint model–harness updates:* WHALE (Kim et al., 2026a), Co-Harness (Chen et al., 2026e).
Target agent's harness	**Editable interfaces**	**AHE** (Lin et al., 2026): seven file-represented component types, with fixed tracing and verification infrastructure. *Related scope:* prompt search in GEPA (Agrawal et al., 2026); module edits in ModularRSI (Wu et al., 2026b).
	Executable programs	**SICA** (Robeyns et al., 2025): rewrites its Python agent code, including tools and orchestration. *Earlier code-space design:* ADAS (Hu et al., 2025).
	Skills and memory	*Skill documents:* SkillOpt (Yang et al., 2026d). *Memory mechanisms:* MemEvolve (Zhang et al., 2026a), MemCodex (Wang and Liu, 2026).
Proposer	**Diagnostic evidence**	**AHE** (Lin et al., 2026): drills down from layered reports to traces and records predicted edit effects. *Retrospective diagnosis:* RHO (Pan et al., 2026).
	Experience access	**Meta-Harness** (Lee et al., 2026b): reads prior candidate code, scores, and raw traces through a filesystem. *Search-time diagnoses and patterns:* MemoHarness (Huang et al., 2026b).
	Changing the proposer	*Trained editor:* Harness-R1 (Shao et al., 2026). *Editable improvement routines:* STOP (Zelikman et al., 2024), Hyperagents (Zhang et al., 2026d).
Evaluator	**Feedback access**	*Task outcomes:* **Meta-Harness** (Lee et al., 2026b). *Self-preference:* RHO (Pan et al., 2026); *unlabeled-task proxies:* TTHE (Nie et al., 2026b).
	Objectives and verification	*Quality, cost, and time:* **SICA** (Robeyns et al., 2025); *safety and utility:* HARDE (Liu et al., 2026e). *Staged checks:* HarnessBank (Luo et al., 2026e); *multi-fidelity:* HARBOR (Sengupta and Wang, 2026).
Controller and history	**Search and retention**	**DGM** (Zhang et al., 2026c): samples an archive of agents to explore alternative improvement lineages. *Workflow tree search:* AFlow (Zhang et al., 2025b); *incumbent refinement:* AIDE2 (Srikanth et al., 2026).
	Admission and integration	*Bounded edits and resource-aware acceptance:* RRSI (Xia et al., 2026). *Statistical gates:* SAGE (Wang et al., 2026d); *recombination:* HarnessBank (Luo et al., 2026e).
	Experiment allocation	*Candidate budgets:* HGM (Wang et al., 2026b); *task selection:* HarnessLens (Xu et al., 2026b). *Feedback-generating curriculum:* ActiveSaddler (Park et al., 2026).
	Timing and meta-updates	*Test-time evolution:* TTHE (Nie et al., 2026b); *coupled phases:* WHALE (Kim et al., 2026a). *Editable evolver policies:* HSI (Zhou, 2026); *task/meta programs:* Hyperagents (Zhang et al., 2026d).

Figure 4: Selected literature across five harness-improvement components. Bold method names identify the four expanded case studies in Appendix A.4; other works illustrate complementary mechanisms and earlier methodological foundations. Braces denote overlapping categories, not temporal succession or an influence ranking. MemoHarness is shown in its search-time role; Table 2 distinguishes deployment reuse. Task-agent memory and optimization history remain distinct; recursive scope is assessed in Section 3.

Method / source	Evidence available	Decision and updated object	Verification and boundary
AFlow (Zhang et al., 2025b) Secs. 3–4	Evaluated code-defined workflows and execution feedback retained in a search tree.	A Monte Carlo tree-search variant selects and expands candidate workflows through LLM-generated changes.	Execute candidates on tasks. A node is a complete workflow, not an action within one task trajectory.
HGM (Wang et al., 2026b) Secs. 3.3, 4.1	Agent–task outcomes aggregated across an ancestor's clade, including evaluated descendants.	Use clade-metaproductivity estimates and Thompson sampling to choose expansion; allocate evaluation at the agent–task level separately from generating children.	The controller estimates the value of continuing an ancestry. This is a prescribed search policy, not evidence that the policy itself evolved; descendant outcomes need not track the parent's current score.
AIDE^2 (Srikanth et al., 2026) Secs. 2.2, 3.5; Figs. 5–6	Research-agent grades select outer-loop rewrites. Within a task, candidate results and error histories inform the discovered agent's search.	The discovered AIDE_{85} replaces greedy solution refinement with strategy-arm selection and periodic forking; it also replaces concatenated history with bounded summaries and conditional failure memory.	These are reported changes inside the research agent. The outer loop still edits the best incumbent; the diverse inner search is not an archive-based outer search.
TTHE (Nie et al., 2026b) Sec. 4	Own-lineage and cross-lineage traces from the current unlabeled task batch; available proxies and judge probes.	Each branch edits its own parent while sharing evidence; a final judge selects among final-round candidates.	Selection lacks target gold labels. The evaluated batch supplies adaptation evidence, so the setting is transductive.

Table 1: Search strategies: evidence, update mechanism, and verification boundary.

A.3 Comparative Design Choices

The comparisons follow Sections 2.2–2.5. Each table records available evidence, the decision and updated object, and the verification boundary. Entries describe published procedures unless explicitly marked as discovered artifacts; the tables compare mechanisms rather than rank performance.

A.3.1 Search Strategies

Table 1 separates workflow-tree expansion, ancestry selection, evolved inner search, and evidence sharing across parallel lineages. HGM estimates the value of an ancestry; AIDE^2 distinguishes the fixed outer optimizer from the search policy discovered inside its research agent.

Alternative retention and search objects. Beyond the main-text comparisons, HyEvo maintains islands of hybrid LLM/code workflows, while MaAS learns a distribution over query-dependent architectures (Xu et al., 2026a; Zhang et al., 2025a). Diversity can therefore reside in retained lineages or in the distribution generating a task-specific architecture. These are different search objects rather than interchangeable versions of one tree policy.

A.3.2 Memory Management

Table 2 distinguishes experience used to propose edits or configure execution from the executable procedures that manage it. The same study may occupy multiple roles or phases; storing more experience, rewriting its retrieval procedure, and changing the improver's retrieval policy are different interventions.

Related task-memory methods. ReasoningBank extracts strategies from successful and failed trajectories; ReMe retrieves, rewrites, and prunes experience; CHIME separates planning and execution memories and attributes credit to guide updates (Ouyang et al., 2026; Cao et al., 2026; Ye et al., 2026c). Their relevance is the experience made available to the target agent. Updating this content is distinct from evolving the procedures that construct, retrieve, and use it, as in MemEvolve and MemCodex (Zhang et al., 2026a; Wang and Liu, 2026).

A.3.3 Guiding Evolution Directions

Table 3 compares diagnosis from existing traces (RHO), acquisition of new failure evidence (ActiveSaddler), and edit-specific checks for improvement and regression (HarnessLens). These mechanisms can coexist: a diagnosed weakness can guide evidence collection and later verification. They differ from selecting the parent lineage itself.

A.3.4 Coupling with Model Training

Table 4 distinguishes training the target model from training the harness editor. It records the source of supervision and what remains fixed, so a gain from joint model–harness updates is not attributed to harness revision alone.

Method / source	Evidence available	Decision and updated object	Verification and boundary
MemoHarness (Huang et al., 2026b) Secs. 2.4–2.6	Search-time experience accumulated as case records and distilled patterns; the resulting bank is reused at deployment.	During search, retrieve evidence to propose global harness updates; at deployment, configure a case-specific harness from the frozen bank.	The bank is frozen at deployment; adaptation uses no new task feedback or additional search. The deployment phase reuses learned experience.
MemEvolve (Zhang et al., 2026a) Secs. 3.2, 4	Inner-loop trajectories and performance under candidate memory architectures.	Diagnose and redesign executable encoding, storage, retrieval, and management modules; select architectures for further evolution.	Evaluate each architecture while its experience base grows from an empty initialization. The outer loop changes memory mechanisms, not only stored content.
MemCodex (Wang and Liu, 2026) Sec. 2.2; Algorithm 2	Failed query traces, lost supporting turns, parent layer code, and a log of rejected edits.	Diagnose the first method losing supporting evidence; each child rewrites one method of one layer or toggles one layer. Accepted architectures enter an archive; rejected edits remain diagnostic records.	Check memory constraints, then compare accuracy and token cost on the same questions with the task agent and judge fixed. Improvements must not worsen either metric; accuracy-only gains require a paired significance test. This is the published update protocol.
MetaSkill-Evolve (Wang et al., 2026f) Secs. 3.3–3.5, 4.3	A persistent graph stores evaluated branch states, lineage and cross-branch inspiration links; recent descendant gains supply a slow-loop signal.	Periodically turn the improvement history into a meta-failure trace and apply the improvement pipeline to branch-specific meta-skill files, including retrieval and proposal policies. Descendants inherit the resulting snapshot.	The changed object guides the improver, not the target agent's task-memory store. Frozen-meta-update and retrieval ablations test these components' roles; they do not isolate every individual file revision.

Table 2: Memory management: evidence, update mechanism, and verification boundary.

Method / source	Evidence available	Decision and updated object	Verification and boundary
RHO (Pan et al., 2026) Sec. 4; App. D	Individual trajectories and recurring deficiencies identified across trajectories, without reference answers.	Retrospective diagnosis produces harness-edit proposals; trajectory self-preference compares candidate behavior.	Retain proposals with positive aggregate preference. This signal is a proxy, not an independent correctness label.
ActiveSaddler (Park et al., 2026) Sec. 4; App. K	A growing registry of failure patterns, supporting scenarios, diagnostic evidence, and optimization history.	Group diagnosed failures into evolving failure-pattern arms under a non-stationary bandit formulation. An LLM reassesses severity, fixability, breadth, and side-effect risk from the latest history; stochastic arm selection and a separate exploration decision govern revisiting known patterns or exploring unseen scenarios.	The harness updater remains unchanged. Earlier successes become eligible for revisiting after coverage conditions, allowing regressions to be rediscovered.
HarnessLens (Xu et al., 2026b) Secs. 4.2–4.3; App. A.4	Successful strategies and recurring deficiencies linked to supporting trajectories, affected components, and task groups.	Propose supported behavior changes; select recovery, positive-control, preservation, and diagnostic tasks around each proposed edit.	Require attributable improvement without observed regression, then confirm on a mostly fresh batch. Checks cover sampled behavior, not all capabilities.

Table 3: Guiding evolution directions: evidence, update mechanism, and verification boundary.

Method / source	Evidence available	Decision and updated object	Verification and boundary
Co-Harness (Chen et al., 2026e) Secs. 3.2–3.5	Diagnosed execution failures and trajectories collected under an improved harness.	Validate harness repairs, then use improved-harness trajectories to fine-tune the task model; alternate the two loops.	Patch validation precedes training-data generation. Model parameters and harness artifacts change through distinct update steps.
WHALE (Kim et al., 2026a) Secs. 3–4	Reward-filtered rollouts for model training; candidate execution scores and traces for harness search.	Alternate rejection-sampling fine-tuning with Meta-Harness search under the updated model.	Each phase holds its counterpart fixed. Joint optimization changes both weights and harness; it need not rewrite the alternating procedure.
Harness-R1 (Shao et al., 2026) Sec. 3	Failure packets and task outcomes from fresh executions with proposed runtime patches.	Train a dedicated harness engineer with supervised fine-tuning and then outcome-grounded GRPO.	Rerun the same task batch with each patch while holding the target fixed during engineer training. The learned object is the editor's policy.

Table 4: Workflow extensions: evidence, update mechanism, and verification boundary.

A.4 Illustrative Revision Histories

Selection and reading guide. These cases examine how evidence becomes a persistent update. Figure 4 maps representative literature to the five workflow components and their design dimensions. Table 5 follows concrete revision histories in SICA, DGM, Meta-Harness, and AHE: the reported problem, the changed artifact, and the observation supporting its interpretation. These cases contrast tool refinement, alternative ancestry, learning from rejected trials, and execution-time enforcement. The preceding Tables 1–4 compare search, memory, direction guidance, and training; this table instead follows concrete before-and-after revisions. Selection reflects these contrasts rather than a citation ranking. Dates denote first public arXiv submission; source locators refer to the cited versions. Empirical observations and our comparative readings are separated; within-run gains are not cross-paper performance rankings. Section 4 and its appendices organize task-specific artifacts more broadly.

Cross-case synthesis. *Current performance and continuation value are different quantities.* SICA's best-parent rule, DGM's productive ancestry through score dips, and HGM's descendant-based estimate expose different ways of spending further search on existing candidates (Robeyns et al., 2025; Zhang et al., 2026c; Wang et al., 2026b). AIDE^2 adds a separate distinction: a discovered inner search policy should not be used to classify its fixed outer optimizer (Srikanth et al., 2026). The separate ignition test and its evidence limits are discussed in Appendix B.

Experience reuse and mechanism revision require different evidence. A frozen MemoHarness bank can configure execution without new search; MemCodex instead evaluates executable memory edits, and MetaSkill-Evolve updates policies that guide future edits (Huang et al., 2026b; Wang and Liu, 2026; Wang et al., 2026f). A comparison must identify the consumer of the experience and the artifact that persists, not simply report that a system has memory.

Traceable changes need not establish isolated causal gains. Meta-Harness's rejected trials explain a change of proposal direction; AHE's passing trace shows an execution guard taking effect. The former is qualitative search evidence, while the latter accompanies a paired intervention (Lee et al., 2026b; Lin et al., 2026). Training introduces another attribution boundary: Co-Harness and WHALE update the task model, whereas Harness-R1 keeps it fixed while training the editor (Chen et al., 2026e; Kim et al., 2026a; Shao et al., 2026). These distinctions determine which part of the improvement process the evidence actually supports.

Case / source	Reported problem and evidence	Concrete update	Observation and interpretation
SICA (Robeyns et al., 2025) 21 Apr 2025 Secs. 3–4; Table 1; App. D	The seed relies on whole-file overwrites, described as slow and token-inefficient. The mixed benchmark includes synthetic file editing and symbol navigation; utility includes performance, time, and cost.	The update sequence introduces SmartEditor and diff/range edits, followed by an AST symbol locator at iteration 9. Later revisions combine lexical candidate search with AST verification.	*Observed:* iteration-level results accompany the reported tool changes; final retention of every listed edit is not established. *Reading:* task feedback can motivate reusable interfaces rather than only better task answers.
DGM (Zhang et al., 2026c) 29 May 2025 Sec. 4.4; Fig. 3; Apps. C.2, F	The seed views and edits whole files. Archived agents inspect benchmark logs before implementing descendants; selection balances performance with the number of functional children.	Discovered changes include line-level viewing, string replacement, repeated solution attempts, and model-based selection among attempts. The archive retains alternative ancestors for later self-modification.	*Observed:* the final best SWE-bench lineage passes through two score dips; the archive-based run outperforms the no-open-ended-exploration baseline. *Reading:* an intermediate score loss need not eliminate a useful ancestry; it does not make every weaker candidate valuable.
Meta-Harness (Lee et al., 2026b) 30 Mar 2026 Apps. A.2, B.3	In the reported TerminalBench-2 search, prompt and completion-flow changes repeatedly regress from Terminus-KIRA. The proposer compares earlier code and traces, identifying a shared prompt intervention across different failed fixes.	After six regressions, iteration 7 adds an environment snapshot before execution, including available tools and files, while preserving the established control loop. Rejected trials inform this change of direction.	*Observed:* the additive candidate becomes the best in that run; the discovered harness includes environment bootstrapping. *Reading:* history can change which intervention is attempted, not merely provide examples to imitate. This is a qualitative trajectory, not a universal preference for additive edits.
AHE (Lin et al., 2026) 28 Apr 2026 Apps. C.1.2, C.2.2; Fig. 9	In `path-tracing`, the agent produces and checks the output, then deletes it during cleanup. Prompt advice against destroying verified state is already present but unenforced; iteration 4 scores 0/2.	The iteration-5 pair adds publish-state rules and a stateful shell guard. The guard records protected outputs after a successful check and intercepts later commands that would delete or reset them.	*Observed:* the next two rollouts pass; traces show the guard blocking cleanup. *Reading:* diagnosis can shift an intervention from advice into execution control. The paired edits and two-rollout observation do not isolate each component's contribution.

Table 5: From observed problems to reported harness changes. Each row follows a concrete revision history. The last column distinguishes the paper's observations from our comparative reading; a before–after result does not isolate every edit's causal effect.

B Configurations and Evidence of Recursive Reuse

This appendix supports Section 3. Appendix B.1 first separates autonomous execution from autonomous updating. Appendix B.2 then asks which changed mechanism is used in subsequent improvement, comparing agents, skills, search policies, and evaluation. Appendix B.3 distinguishes a proposed mechanism, demonstrated reuse, and evidence of benefit. These are complementary comparisons within the main text's five-level framework.

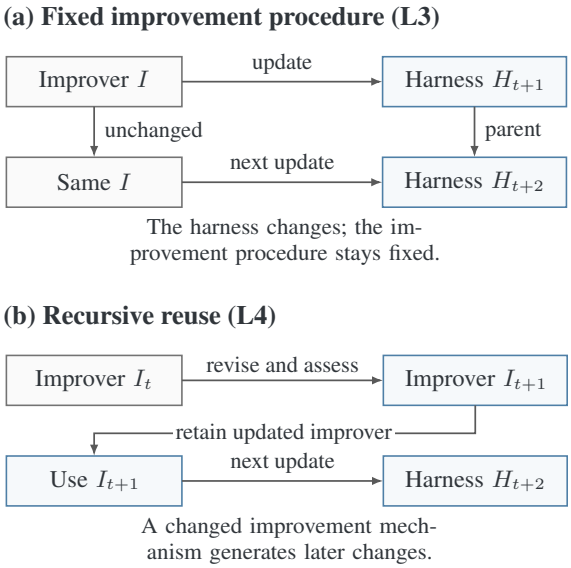


Figure 5: Fixed versus recursively reused improvement mechanisms. In (a), the same improver I produces successive harness revisions. In (b), an updated I_{t+1} is retained and invoked for later modification. This distinguishes L4 from L3 under Liu et al. (2026d); fixed outer constraints may remain. The schematics establish neither measured gains nor general recursion.

B.1 Update Autonomy

Human-assisted configurations. Continual Harness contrasts human-guided revisions with an automated refiner. ARIA identifies knowledge gaps and requests corrections from human experts before updating persistent knowledge; Growing with Your Embodied Agent converts human corrections into reusable manipulation skills (Karten et al., 2026; He et al., 2025; Meng et al., 2025). These configurations preserve the distinction between autonomous task execution and the human input required for substantive improvement. Their inclusion does not imply that the mechanism producing later updates itself evolves.

B.2 Mechanisms Reused in Later Improvement

The comparison follows the operation performed by the updated artifact. A system can change several such operations; the groups below are not additional RSI levels.

B.2.1 Agents and Improvers

STOP rewrites an improver program, while SICA and DGM reuse modified coding agents to implement subsequent changes (Zelikman et al., 2024; Robeyns et al., 2025; Zhang et al., 2026c). Hyperagents makes meta-level modification explicit, and Multi-Task Self-Harness uses the evolving harness for both solving and proposing (Zhang et al., 2026d; Xu, 2026). AIDE^2 shows why configurations must be separated: its main outer loop edits an inner research agent, whereas a separate *ignition test* deploys an evolved agent as the outer improver. That test demonstrates reuse in the improving role, but its three seeds per arm do not establish an efficiency advantage over the original outer agent (Srikanth et al., 2026, Sections 2 and 3.6).

B.2.2 Improvement Skills and Procedures

The distinction is who invokes a retained skill. MetaSkill-Evolve updates procedures for analysis, retrieval, allocation, proposal, and editing using descendant outcomes; these procedures guide later improvement (Wang et al., 2026f). Promptbreeder similarly evolves mutation prompts used to generate later task-prompt changes (Fernando et al., 2024). By contrast, persistent task-execution skills, such as those produced by HASP, need not change the procedure generating and validating skills (Liu et al., 2026a). Calling both artifacts "skills" conceals different recursive roles.

B.2.3 Selection and Search Policies

HSI rewrites evolver strategies, while Dream-RSI improves an exploration policy through replay and redeploys it for later search (Zhou, 2026; Zheng et al., 2026). Their revised policies influence subsequent optimization, although the outer training or meta-evolution protocol remains prescribed. HGM's clade-based selection and MILO's adaptive orchestration provide useful contrasts (Wang et al., 2026b; Jana et al., 2026): different decisions in response to history do not alone show that the decision rule itself evolved. The search comparisons in Section 2.2 therefore do not automatically imply different RSI levels.

B.2.4 Evaluation and Feedback Mechanisms

The Last Harness You'll Ever Build proposes a meta-evolution loop over a blueprint containing the evaluator and evolution agent (Seong et al., 2026). This conceptual framework broadens the proposed editable mechanism; it does not itself establish an experimentally retained evaluator revision. Evolving tasks or environments, as in Env-Rethink, instead changes the experience supplied to an optimizer; recursion in the generator requires evidence that its own update mechanism changes and is reused (Wu et al., 2026c). When evaluation becomes editable, progress also needs an independent criterion so that changes in the judge are distinguishable from improvements in the agent.

B.3 Interpreting the Evidence

The cases distinguish three evidential steps: specifying an editable mechanism, showing that its changed version is invoked in later improvement, and measuring the benefit of that reuse. A proposed blueprint, an executed revision, and a matched comparison answer different questions. This distinction connects the mechanisms above to the task-level modifications in Appendices C–F.

C Coding Harness Evolution: Supplementary Evidence

This appendix supports Section 4.1: program generation, repository engineering, terminal tasks, and ML engineering. Source locators appear in Appendix C.5.

Coverage and organization. Four task categories contain 36 category–study entries covering 38 study–setting records from 26 studies. Related benchmarks are kept within one paper's category entry; a paper appearing in several categories or transfer settings is not an independent replication. This is a selected evidence set, not an exhaustive census.

Evidence scope. Each entry states the starting setting and seed, reported reusable change, and evidence and limits. Retained executable changes, intermediate discoveries, skill-bank examples and update procedures remain distinct. A higher benchmark score alone does not identify what evolved, and whole-harness gains do not isolate constituent mechanisms. Frozen transfer, test-time adaptation and transfer of an updating skill manager are identified separately.

Figure 6 summarizes representative seed structures. The four task families below organize the evidence for the interface, context-access, and failure-recovery patterns discussed in the main text.

C.1 Program Generation and Editing

The comparison follows how program-generation harnesses edit structured code, select candidate programs, and respond to executable checks.

SICA: local editing and symbol navigation. (Robeyns et al., 2025) [S24]

Setting and seed. LiveCodeBench participates in one mixed-task evolution run. The seed already has subagents and file-context management but modifies files only through whole-file overwrites.

Reported evolution. Whole-file overwrites are extended with diff- and range-based editing, followed by a Python AST symbol locator in iteration 9. Appendix D's update sequence also reports a SmartEditor InputNormalizer (update 6), reduced diff-context output (8 and 13), MainAgent detection and routing of editing tasks (10), and AST parsing restricted to relevant code regions instead of entire files (14). These change the reusable editing interface and the amount of code processed or returned.

Evidence and limits. These are reported implemented updates; retention in the final utility-winning agent remains unconfirmed. The normalizer's exact transformations were not independently checked in code. The 15-round run combines 50 SWE tasks, 50 LiveCodeBench tasks and synthetic editing/navigation tasks; synthetic navigation mainly motivates the AST tool. No individual ablation establishes the effect of these updates, and their appearance in two benchmark records is not independent replication. The seed's XML tool-call format, subagents and context management are inherited capabilities.

TTHE: verification only after exposed failures. (Nie et al., 2026b) [S25]

Setting and seed. Test-time adaptation uses a recent hard subset of 60 LiveCodeBench problems, starting from a ReAct scaffold without the reported conditional verification-and-repair route; the release window is unspecified.

Reported evolution. The selected scaffold retains a low-cost solving path and adds verification and repair when public tests fail. Additional execution is thus conditional on an observed failure, rather than required for every problem.

Evidence and limits. The route is reported as retained. Adaptation and scoring use the same batch, making this a transductive result rather than a sealed generalization test. There is no individual-change ablation. The SWE empty-patch retry and DS-1000 tasks in the shared configuration do not establish additional LiveCodeBench mechanisms.

Darwin Gödel Machine: recoverable edits and candidate delivery. (Zhang et al., 2026c) [S22] [S34]

Setting and seed. Aider Polyglot evolution starts from bash, a whole-file editor and one agent attempt. Optimization uses 60 problems; 165 further problems support generalization evaluation.



Figure 6: Coding task families, evaluation settings, and seed-harness structures. Italics identify concrete seeds or study-specific initial configurations. Bold labels summarize starting architectures and supported variants within each family; individual seeds need not contain every listed capability. Braces group benchmarks and structures at the family level. Study-specific configurations and evidence are in Appendix C; transfer settings refer to the source lineage's seed. Families overlap, Terminal-Bench versions remain distinct, and EvoAgentBench LiveCode is separate from LiveCodeBench.

Reported evolution. The retained implementation adds replacement, insertion, range editing and undo, plus up to three attempts with workspace resets. Patch/test summaries inform later attempts; execution success and test counts guide selection, after which the selected patch is replayed. Local revision becomes recoverable, while candidate exploration is separated from final delivery.

Evidence and limits. The full-benchmark score increases from 14.2% to 30.7% (Section 4.4). This includes the optimization subset and measures the complete lineage, not either mechanism in isolation. Appendix F.2 gives the retained code, but its diff has not been linked to a repository commit. Internal tests do not access private ground truth. The coarse seed interface is a documented limitation, not an isolated failure-trace experiment.

EvolveNet: separating library-specific and shared behavior. (Nie et al., 2026a) [S29]

Setting and seed. DS-1000 evolution starts from a 17-line shared seed program, before client-specific improvements are merged.

Reported evolution. The round-2 example keeps schema preanalysis inside the Pandas branch while moving broadly applicable execution retries into shared cross-library logic. The merge preserves the intended scope of each client specialization.

Evidence and limits. This is a reported merge discovery, not independently inspected final source; final retention remains unconfirmed. The protocol uses local feedback, server validation and held-out tests, but supplies no causal ablation of this merge. The Pandas-specific example does not establish a concrete discovery on SWE or LiveCodeBench, nor does a short seed alone identify a particular tool architecture.

AFlow: composing generation, selection and conditional repair. (Zhang et al., 2025b) [S53]

Setting and seed. MBPP search begins with an empty workflow template and manually predefined operators. The LLM revises reusable code and prompts using parent workflows, modification history and execution feedback.

Reported evolution. The selected round-14 program generates three candidates, selects with ScEnsemble and invokes Test. Failed tests route their output to FIX_CODE_PROMPT for repair. The evolved object is this executable composition and its conditional branch, rather than a newly invented operator or a single task answer.

Evidence and limits. Each candidate receives five validation runs, and the cited listing is the retained best MBPP program. The failure branch passes the problem, failed solution and returned test error to the repair prompt; the listing returns the repaired candidate without a further test call. A separate test-generation example is not this final chain. Component-level causal gains are not established, and the workflow must not be relabeled as HumanEval's selected implementation.

HarnessBank: recovery after runaway reasoning. (Luo et al., 2026e) [S20]

Setting and seed. The EvoAgentBench LiveCode setting starts from a vanilla harness that can return an empty turn after reasoning exceeds its budget. The precise initial implementation is unspecified.

Reported evolution. A retained recovery branch disables thinking only after runaway reasoning prevents output from resuming. Ordinary execution keeps its original reasoning mode, so the change is event-triggered recovery rather than globally disabling thinking or uniformly expanding the budget.

Evidence and limits. Training and test data are separate, with one sealed test after training-based selection. The relationship between EvoAgentBench LiveCode and official LiveCodeBench, including release alignment, remains unverified. The recovery example cannot be assigned to official LiveCodeBench or Terminal-Bench, and it does not isolate the causal contribution of the branch from the complete harness.

MermaidFlow: inheriting testing and candidate diversity. (Zheng et al., 2025b, Section 5.4, Figure 4; Appendix B.3)

Setting and seed. HumanEval search represents workflows as editable Mermaid graphs over supplied operators. The illustrated crossover starts from two existing parent workflows, not a newly identified universal seed.

Reported evolution. Workflow_8 combines the test node introduced by Workflow_4 with Workflow_5's more diverse ensemble. Verification and candidate diversity can therefore be inherited from different parents through executable graph composition.

Evidence and limits. Section 5.4 and Figure 4 document this generated child. Appendix B.3 separately provides a selected HumanEval implementation; the crossover child is not automatically that final winner. The reported score does not isolate the testing node's effect, and the operators themselves are supplied rather than evolved by this example.

FlowEvo: reusing skills directly or as construction context. (Ren et al., 2026, Sections 3–4; Appendix A)

Setting and seed. HumanEval and MBPP use generated workflows augmented by a persistent skill bank. Successful executions can supply procedures for subsequent tasks; Appendix A separates bank growth from frozen-bank MBPP evaluation.

Reported evolution. Acquired skills can execute directly when compatible, or provide structured context for constructing a new workflow. This changes what is retained between problems and how the next solver uses it.

Evidence and limits. On the 71 exact-transfer cases, direct reuse reaches 85.9% versus 81.7% for dynamic generation with lower token use; text memory reaches 87.3% at higher cost. No router pair separates significantly on the full evaluation split. These comparisons support conditional reuse, not universal replay superiority or a final coding-specific skill lineage. The detailed ALFWorld listing is not a coding artifact.

C.2 Repository-Level Software Engineering

The comparison follows repository navigation, patch construction, and the checks needed to establish that a patch works in its environment.

Darwin Gödel Machine: editing, patch selection and context recovery. (Zhang et al., 2026c) [S22] [S23]

Setting and seed. SWE-bench evolution begins with a fixed model, bash and whole-file editing. Test execution and test summarization are already seed capabilities.

Reported evolution. The best lineage adds line-range viewing and unique string replacement. Appendix F.1's added `is_patch_valid` rejects empty or whitespace-only patches, patches with no parsed modified files, and changes classified as test-only by a filename heuristic. The execution loop resets the workspace and supplies the failure reason before another attempt, with up to three attempts in total. It also compares candidates with internal tests and summarizes older messages after context overflow before resuming. Testing itself is inherited; the evolved changes govern its use and the recovery paths.

Evidence and limits. The changes are reported as retained. Screening on 10+50 tasks precedes evaluation on 200 tasks that include those screening tasks; this is not an independent 500-task test. No individual causal ablations apportion the overall gain among the changes. Context summarization does not guarantee preservation of every earlier tool result.

SICA: extending the inherited editing interface. (Robeyns et al., 2025) [S24]

Setting and seed. SWE-bench Verified shares one mixed-task run with LiveCodeBench and synthetic tasks. The seed already contains subagents, an asynchronous overseer and file-context management, but editing is limited to overwriting files.

Reported evolution. Evolution adds diff- and range-based editing, followed by an AST symbol locator in iteration 9. Appendix D further records editor-input normalization, task-based edit routing, smaller

diff-context returns and parsing only relevant AST regions; the program-generation entry gives the update numbers. The existing subagents, overseer and XML tool-call parser are inherited architecture.

Evidence and limits. The fixed 50-task SWE subset participates in a 15-iteration mixed-task run, not an independent repository-only search. These are reported implemented discoveries whose retention in the final utility winner has not been verified from its diff. Overall SWE improvements cannot be assigned to individual changes, and the AST discovery is primarily associated with synthetic navigation tasks.

Multi-task Self-Harness: one lineage across three repository settings. (Xu, 2026) [S01]

Setting and seed. The seed is a 49-line prompt-and-bash loop that appends raw outputs without timeouts, compression or recovery. SWE-Bench Pro contributes 60 evolution tasks and 100 held-out tasks from a pool of 731. Verified (100 tasks) and DeepSWE (53 tasks) are frozen out-of-distribution evaluations, excluded from evolution.

Reported evolution. Outputs previously appended verbatim are bounded: above 48,000 characters, the harness preserves the beginning and end and inserts a notice suggesting a narrower command or saving output to a file. Outputs older than the latest six are shortened to at most 6,000 characters. These rules appear in iteration 1 and remain in the selected iteration 5. Separate history compaction preserves opening and recent messages, and a fresh-context completion review checks the task against the filesystem.

Evidence and limits. Section 5.2 reports that the seed aborts on 12 SWE-Bench Pro tasks because test or build output exceeds the input limit; iteration 5 solves seven of them. This is a complete-version comparison, not an ablation of truncation. The same lineage serves all three settings, so they are not independent discoveries. History compaction rarely triggers in coding; it is distinct from per-output trimming. The iteration-6 budget cap is excluded. DeepSWE currently has no second extracted study supplying a concrete change for comparison.

Self-Harness: model-specific patch verification. (Zhang et al., 2026b) [S10] [S11]

Setting and seed. Three independent model-specific searches on a 100-task SWE-bench Verified subset start from a minimal DeepAgent harness. Seed MiniMax and Qwen traces make changes but stop without completing executable verification.

Reported evolution. MiniMax separates empty-diff inspection from targeted testing through two subagents. Qwen combines a patch verifier with pre-submission diff/test instructions. In GLM, `build_verification_instruction` previously requested test execution and debugging on failure. The evolved instructions distinguish missing imports or dependencies, ask the agent to install the missing package and rerun the exact test, and resume source debugging only if it still fails. This is a retained instruction change, not a programmatic gate enforcing that sequence.

Evidence and limits. The modifications are retained in separate promoted harnesses. Appendix B.2 reports successful verification in the paired MiniMax and Qwen traces. Its GLM Astropy case (Figure 16) moves from syntax-only checking to dependency repair and executable verification: all four previously failing FAIL-TO-PASS tests then pass. These are whole-harness case comparisons, not ablations of particular instructions or subagents. Table 1 separately gives MiniMax overall Pass of 46.0% versus 52.5%. Both 67 held-in and 33 held-out tasks may affect acceptance, so the latter are not an untouched test set. These are separate searches within one paper, not frozen transfers from Terminal-Bench.

TTHE: treating an empty patch as a rerun condition. (Nie et al., 2026b) [S25]

Setting and seed. Test-time adaptation on a 40-task hard SWE-bench Verified slice begins with stock mini-swe-agent, whose first attempt can terminate with an empty patch.

Reported evolution. The adapted scaffold reruns an empty-patch attempt with explicit reproduce–localize–repair–retest instructions and increases the step budget during multiround search. The change converts an otherwise terminal outcome into a conditional recovery path.

Evidence and limits. The selected mechanism is reported as retained. Selection without gold labels and subsequent scoring use the same batch, so this is transductive adaptation rather than unseen-task

generalization. The budget and instructions change together. LiveCodeBench's public-test mechanism is not assigned to SWE, and the example's small-patch constraint is not general repository-repair advice.

HarnessCompass: accumulating conditional repair guidance. (Zhang et al., 2026e) [S26]

Setting and seed. Search on SWE-bench Verified starts from a short prompt and shell tool, without memory, middleware, skills or subagents. The split contains 50 search and 450 evaluation tasks, with additional cross-model evaluation.

Reported evolution. Retained memory adds conditional guidance for identifying responsibility along execution paths and covering affected callers when changing shared interfaces. An ungated control instead accumulates more task-specific repair records.

Evidence and limits. The examples are reusable guidance, not enforced tool behavior. They support a distinction between conditional repair knowledge and memorized fixes, but neither rule has an independent ablation. Gains from the complete structure and guidance tracks cannot be attributed to memory alone. Evaluation on unseen tasks does not make the two rules independent discoveries.

AHE: transferring execution and delivery controls. (Lin et al., 2026) [S04] [S05]

Setting and seed. All 500 SWE-bench Verified tasks evaluate frozen transfer from Terminal-Bench 2 evolution. The source seed, NexAU_0, exposes only bash and has no middleware, skills or long-term memory.

Reported evolution. The transferred harness exposes per-call timeouts, protects accepted artifacts from destructive actions, forwards cross-step risk reminders to the next model turn, and checks acceptance criteria before completion. These occupy distinct tool, interception, context and completion modules; reminders are not equivalent to execution-level blocking.

Evidence and limits. The changes are reported as retained, but every concrete edit was discovered in the terminal source domain. They are not SWE discoveries. One lineage introduces interacting components, and no isolated SWE contribution is established for any single mechanism. Frozen transfer therefore supports reuse of the complete harness, not independent evolution on another benchmark.

DarwinX: transferring artifact and acceptance guidance. (Zhang et al., 2026g) [S14]

Setting and seed. SWE-bench Verified is a 500-task frozen transfer target after skill evolution on Terminal-Bench 2.1. The source seed is base Monet without the seven later verification/artifact skills.

Reported evolution. The retained bundle extracts acceptance contracts, grounds deliverables in real tool results, checks and repairs artifacts, repeats checks after modifications, and repairs safety/interface-contract violations before completion.

Evidence and limits. The seven skills are jointly selected execution guidance, not separately ablated mechanisms or an unavoidable submission blocker. The bundle's SWE evaluation does not show where the skills were discovered. Terminal-Bench, TerminalWorld and different model configurations have distinct protocols; evidence from those configurations must not be merged into a single repository evolution run.

Mixture of Self-Improving Branches: retrieval before the first action. (Dong et al., 2026) [S12] [S13]

Setting and seed. SWE-bench Lite uses mini-swe-agent as its original baseline. The immediate parent of the illustrated modification already has structured editing; the 300 tasks are split into 50 development and 250 test tasks.

Reported evolution. The candidate extracts up to eight issue identifiers, ranks files by matches, reads two source files and one test file, and injects bounded content into the initial message. This changes initialization through targeted retrieval, rather than adding the parent's existing unique-match replacement tool.

Evidence and limits. Appendix E.4 supplies a discovery example; retention in the final deployed head is unverified. Development-selected heads and test-best candidates are reported separately and must not be conflated. The example does not establish an individual causal gain or a universal best retrieval budget.

RHO: connecting requirements to executable verification. (Pan et al., 2026) [S15] [S16]

Setting and seed. SWE-Bench Pro uses a domain-specific optimization from vanilla Codex, before its repair-verify helper and requirements ledger. It is not a frozen transfer of the terminal configuration.

Reported evolution. A retained callable helper finds toolchains outside PATH, runs lightweight language checks, reports generated artifacts and accepts custom verification commands. A separate requirements ledger is revisited before submission.

Evidence and limits. The setup draws 100 training and 100 held-out tasks from the ScaleAI test set using evaluator revision 0c64e26f00b9. Held-out trajectories show helper calls, distinguishing this configuration from the unexecuted terminal scripts. No single-change ablation isolates the helper, and terminal playbook findings cannot be substituted for these repository observations.

OneRecipe: timed reminders for language-specific verification. (Yang et al., 2026c) [S52]

Setting and seed. Multi-SWE-Bench evolution runs by language and model from a minimal mini-SWE plugin: a short prompt, bash, and empty hooks and memory. Full mini-SWE is a separate baseline; evaluation also considers cross-language transfer.

Reported evolution. The retained C++/Haiku example uses before_query to detect recent testing or submission cues and inject compilation/testing reminders. Prompts and memory constrain patch scope, discourage test-file edits and request targeted verification.

Evidence and limits. The procedure uses three rounds, 80 steps and USD 1.5 per rollout. The hook nudges rather than blocks submission. make/ctest depend on the language ecosystem, and the test-file restriction reflects the hidden gold-test-patch convention. Aggregate transfer results do not isolate the contribution of either guidance content or trigger timing.

CODESKILL: task procedures and event-triggered diagnostics. (Li et al., 2026a, Sections 3–4; Appendix D, Figures 4–5)

Setting and seed. A separately trained manager maintains natural-language skills around frozen coding policies. EnvBench uses a ReAct-style agent and SWE-bench Verified uses mini-SWE-agent; both expose bash-based actions.

Reported evolution. The published merged Maven task-level skill distinguishes environment, configuration and dependency failures before code edits. Its merged event-level counterpart diagnoses a missing test-jar, checks its producing module, arranges generation/installation, and verifies the artifact before retrying the build. Persistent updates change skill instructions, not the supplied tools.

Evidence and limits. These are bank examples rather than individually scored skill ablations. Project build-file edits are downstream actions, not evolved harness code. The test tasks are held out from manager-policy training, but Section 4.1 constructs their skill bank in two preparatory passes: extraction from no-skill runs, then evolution from skill-conditioned runs. Final evaluation filters out skills whose recorded provenance includes the current instance. This is test-set bank preparation, not evaluation of an unchanged training-only bank; benchmark gains do not isolate either Maven example.

GSE: generalizing diagnostic skills across repositories. (Yang et al., 2026a, Sections 4–5; Figure 3)

Setting and seed. OpenHands and mini-SWE-agent are evaluated on bug-revealing test generation for 108 Java bugs derived from Multi-SWE-Bench, and false-positive filtering for 500 IndustrialBugs reports. These settings evaluate defect analysis, not patch resolution.

Reported evolution. Figure 3 consolidates narrow collection-size and permission rules into a skill distinguishing value-triggered from state-triggered defects, linked to existing test-construction skills. The filtering case replaces conflicting project-specific nullability rules with reasoning about nullability guarantees and propagates that abstraction to related skills.

Evidence and limits. One-project-held-out evaluation tests reuse across repositories. The examples illustrate learned abstraction and compatibility between skills. Aggregate results and architecture-level

ablations do not isolate either example's causal effect, and the simplified Figure 3 illustration should not be treated as a full final-bank artifact.

SkillForge: acquiring repository-specific repair knowledge. (Chen et al., 2026c, Section IV-E; Figure 6)

Setting and seed. Before real issues are presented, synthetic issue solving acquires project-specific skills associated with repository entities. The downstream repair agent retrieves this knowledge rather than rewriting its controller during the illustrated repair.

Reported evolution. In Django-11206, the acquired guidance steers the agent away from an exponent heuristic toward the existing Decimal pipeline, precision-dependent numerical equivalence and boundary-case checks.

Evidence and limits. The paired traces in Section IV-E pass 0/2 and 2/2 fail-to-pass tests without and with skills. This demonstrates an application of acquired knowledge, not a separate causal estimate for each instruction. The persistent object is entity-grounded guidance; the particular code patch is a task output. Knowledge acquisition and downstream repair therefore remain distinct stages.

C.3 Terminal-Based Engineering Tasks

The comparison follows stateful command execution, selective exposure of files and logs, and checks that preserve the required deliverable.

Meta-Harness: discovering the environment before reasoning. (Lee et al., 2026b) [S03]

Setting and seed. Terminal-Bench 2.0 search and evaluation share 89 tasks. Search considers Terminus 2 and Terminus-KIRA; the illustrated discovery builds on KIRA, inheriting native tool calling, a 30KB output cap and a completion checklist.

Reported evolution. The retained bootstrap gathers directory contents, language/package-manager versions and memory before the first model call, then injects a snapshot into the initial context. A 15-second timeout and silent failure keep the startup probe from blocking execution.

Evidence and limits. Appendix B.3 reports gains on seven tasks relative to KIRA, especially where specialized tools are needed, and identifies two to four initial environment-discovery turns as the bottleneck. This within-benchmark comparison supplies no independent coding holdout and is not an isolated causal ablation. The inherited checklist and output cap are not new discoveries; the public author artifact identifies version 2.0.

AHE: preserving validated delivery state. (Lin et al., 2026) [S04] [S05]

Setting and seed. Terminal-Bench 2.0 evolution uses 89 tasks for ten iterations. NexAU_0 begins with bash, a default five-minute shell timeout, and no middleware, skills or long-term memory; SWE is reserved for frozen transfer.

Reported evolution. Retained edits expose command-specific timeouts, protect validated artifacts, detect cross-step risks, inject next-turn reminders and check completion. In configure-git-webserver, an intermediate overrideable guard still permits cleanup to erase a working deployment. Paired iteration-8 updates harden the guard and make warnings salient before the next action.

Evidence and limits. That task changes from 0/2 to 2/2 successful rollouts, with destructive cleanup blocked and the live state submitted. This supports the reported mechanism without separating the paired edits. Manifests and commits identify changes, but the historical run has not been reproduced. Components interact non-additively, and Section 4.4.1 reports costs from redundant checks on long-horizon tasks. The current official configuration corroborates version 2.0.

ModularRSI: correcting tool dispatch and completion state. (Wu et al., 2026b) [S08] [S09]

Setting and seed. An initial modular harness evolves on related tasks outside Terminal-Bench, then undergoes frozen evaluation on version 2.0. The separate Section 6.4 method comparison specifies Terminus-2 and does not establish that seed for every main experiment.

Reported evolution. In Appendix D.4's Go repair case, the initial dispatcher rejects multiline `write_file` input and sends it to the shell, producing a command-not-found error. The reported revision identifies `write_file` or `edit_block` by command name and sends the remaining payload to the editing helper. The paper reports promotion in generation 9 and unchanged multiline handling in generation 12. A separate loop update re-presents pending acceptance conditions when the agent signals completion without commands.

Evidence and limits. On the same Go task, three runs per version all succeed; their interaction counts (called episodes in the source) change from $[41, 24, 43]$ to $[26, 18, 31]$, reducing the mean from 36 to 25. These are external evolution-task observations, not Terminal-Bench test cases or an isolated dispatcher ablation: parser recovery and other changes occur concurrently. A source check of the released merged module finds the editing-command branch nested under a `cat`-only condition, making that branch unreachable for the editing names in this snapshot. This is not the historical generation 9/12 artifact; those source snapshots and original rollouts remain unchecked. The current code therefore does not independently confirm the paper-reported fix or invalidate its historical observations.

Self-Harness: model-specific artifact recovery. (Zhang et al., 2026b) [S10] [S11]

Setting and seed. Separate model searches use 64 filtered Terminal-Bench 2.0 tasks. The minimal DeepAgent seed provides read, write, edit and shell tools plus the default task prompt, without the later runtime limits.

Reported evolution. MiniMax's final harness encourages early creation of required artifacts and redirects stalled work through a total tool-message cap. Qwen instead adds dependency/artifact checks, retry discipline, loop recovery and middleware triggered by tool errors.

Evidence and limits. These are retained changes in different promoted harnesses, not one shared patch. Held-out trajectories are withheld from diagnosis, but held-out scores participate in the promotion regression gate, so evaluation is not sealed. This entry records only the extracted MiniMax and Qwen terminal changes; GLM's repository dependency-recovery mechanism must not be transferred to this setting.

Mixture of Self-Improving Branches: checking completion once. (Dong et al., 2026) [S12] [S13]

Setting and seed. Terminal-Bench 2.0 uses 88 tasks split into 30 development and 58 test tasks. The illustrated candidate's immediate parent is verification-enhanced Terminus-KIRA with an existing tool-execution loop.

Reported evolution. The first completion signal triggers a verification prompt and sets a one-time flag. Subsequent tool use leaves the flag intact, so a later completion exits without reopening the same verification request.

Evidence and limits. Appendix E.3 documents a discovered candidate, not a confirmed final head deployed by the router. Its retained state flag bounds repeated checking, but final retention and an isolated effect are unverified. Test-best candidates in the discovery pool must remain distinct from development-selected results. The parent's verification capabilities are inherited rather than new mechanisms.

Multi-task Self-Harness: bounded observations and fresh-context review. (Xu, 2026) [S01]

Setting and seed. Mixed-task evolution includes 60 Terminal-Bench 2.1 tasks, with 29 held out. The seed is a 49-line prompt-and-bash Python loop that accumulates outputs verbatim without compaction or recovery.

Reported evolution. Iteration 5 retains the beginning and end of long outputs, further shortens old observations, and trims overlong history to its opening and recent messages. Instructions allow recovery

of omitted information from files. The first stop decision opens a fresh-context review of the original task and current filesystem.

Evidence and limits. These changes are reported as retained and reused across tasks, not rediscovered independently for each benchmark. History trimming rarely triggers in coding, and candidates are not screened by validation score. The 150-second/eight-call cap added at iteration 6 is excluded from this iteration-5 account; no individual-change ablation is established.

RRSI: bounding audits and waiting behavior. (Xia et al., 2026) [S06] [S07]

Setting and seed. Evolution uses all 89 Terminal-Bench 2.1 tasks, beginning with Terminus-2 before the reported bounded audit and polling guidance. This record concerns candidate R0-A only.

Reported evolution. R0-A adds a bounded completion audit and nonblocking polling guidance for long-running work. Both are jointly accepted at that iteration, modifying how the agent checks completion and waits for results.

Evidence and limits. Acceptance at R0 is documented, but survival in the final harness is unknown. The polling mechanism is guidance, not evidence of a scheduler rewrite. Rejected candidates are not included among the evolved features. SWE does not participate in selection, and this within-benchmark acceptance does not establish held-out generalization or separate effects for the two instructions.

DarwinX: making delivery requirements explicit in skills. (Zhang et al., 2026g) [S14]

Setting and seed. Evolution and evaluation use the same 89 Terminal-Bench 2.1 tasks. Base Monet starts without the seven later verification/artifact-contract skills.

Reported evolution. The retained bundle extracts acceptance conditions, checks artifact paths, formats and values, repairs failures and rechecks after changes. It also requires real tool evidence and addresses safety/interface constraints before completion.

Evidence and limits. The authors describe seven jointly selected skills, not per-skill causal ablations. Their content guides execution rather than demonstrating unavoidable submission gates. TerminalWorld uses a separate train/test protocol, and SWE is a transfer target; neither supplies additional evolution evidence for this terminal entry. Whole-bundle results therefore cannot be assigned independently to each instruction.

RHO: organizing terminal work through a requirements playbook. (Pan et al., 2026) [S15] [S16]

Setting and seed. A terminal-specific optimization uses 30 training and 59 held-out tasks, starting from vanilla Codex with GPT-5.5 high in a configurable workspace. The paper names Terminal-Bench 2 and pins commit 53ff2b87d621 without an established minor-version mapping.

Reported evolution. A retained playbook records task requirements, organizes host-to-container commands, checks final artifacts at their container paths and revisits the specification before completion. Scripts are generated alongside it.

Evidence and limits. Behavioral analysis explicitly reports that terminal trajectories do not run those scripts; the operative observed change is guidance. SWE-Bench Pro's executed repair-verify helper must not be assigned here. Cross-benchmark reuse and an individual causal effect are unproven. This setting remains version 2.x, minor version unspecified, rather than being silently assigned to 2.0 or 2.1.

CODESKILL: transferring a skill-management policy. (Li et al., 2026a, Sections 4.1, 4.3; Appendix C–D)

Setting and seed. Terminal-Bench 2 is held out from skill-manager policy training and evaluated through mini-SWE-agent with frozen downstream model weights. The paper does not establish a minor version for this record.

Reported evolution. The learned manager extracts task-level and event-driven instructions and maintains the bank through additions, revisions, merges and removals. Section 4.1 specifies two preparatory passes

over the test set before final evaluation: initial extraction, followed by skill-conditioned evolution. During final evaluation, skills whose tracked provenance includes the current task are excluded. Transfer therefore concerns the manager with test-set bank preparation; it is neither a training-only frozen bank nor a demonstrated streaming update after each final-evaluation query.

Evidence and limits. This supports terminal evaluation coverage and an update mechanism. The published Maven examples demonstrate skill granularity but are not tied to particular terminal tasks or a terminal-specific final harness. Accordingly, no terminal-specific content or individual mechanism gain is inferred from them; policy training, bank evolution and task execution remain separate processes.

C.4 Machine Learning Engineering

The comparison follows reusable experiment search, execution interfaces, and research guidance; an individual trained model is not itself a harness revision.

AIDE^2: search diversity and bounded experiment history. (Srikanth et al., 2026) [S36]

Setting and seed. MLE-bench Lite22 is a held-out transfer evaluation of AIDE85, evolved on external tasks. Its AIDE0 ancestor greedily selects the best candidate and accumulates full candidate histories during drafting and improvement.

Reported evolution. The retained runtime searches multiple strategy arms with periodic cross-arm forks. It summarizes initial and recent solutions, adds selected error signatures when bugs recur, catches nearly empty code and prompts generalization to unseen test data. Search can change strategy while retaining a strong candidate.

Evidence and limits. Mean private percentile rises from 0.678 ± 0.006 to 0.722 ± 0.011; errors are one standard error. End-of-run median prompt-size reduction reaches $7\times$. These are whole-runtime transfer results, not module ablations, and AIDE47 scores higher on this benchmark (0.730 ± 0.005). Lite22 never enters evolutionary selection. The ineffective ranking penalty, rejected ensembles and scoring-script repair are not retained task-runtime mechanisms in this account.

SIA: turning denoising experiments into structured feedback. (Hebbar et al., 2026) [S71]

Setting and seed. The MAGIC denoising hyperparameter task starts from a Meta-Agent-generated system prompt, single-tool dispatch loop and output parser around gpt-oss-120b. It is task-specific harness iteration, not MLE-bench evaluation.

Reported evolution. With model weights fixed, the Feedback-Agent adds a batched configuration driver and a parser exposing parameter–score pairs. These reusable tools organize experiments and their feedback; particular neighbor counts, diffusion steps, bandwidths and preprocessing choices are task-level candidates rather than harness changes.

Evidence and limits. Section 7.2 reports the discoveries but provides no final-version diff or individual-tool ablation, leaving final retention unconfirmed. Table 2 specifies no separate train/test split for this task. Later GRPO updates and postprocessing belong to the weight-update phase and are excluded from this harness-only record.

AIBuildAI-2: carrying experiment takeaways into future runs. (Zhang et al., 2026f, Figure 1; Methods: Knowledge builder agents and Knowledge system evolution)

Setting and seed. MLE-bench and live competitions use a hierarchical ML agent initialized with a human-authored topical index and roughly 1,000 knowledge documents in about 30 categories.

Reported evolution. After a task, an L2 builder turns the working directory into a categorized takeaway document. An L1 builder revises the category's instruction and its pointers to L2 documents; the top-level category index remains fixed. A second stream incorporates external material, and future executions retrieve relevant guidance selectively.

Evidence and limits. The paper specifies this persistent-update procedure but does not expose a complete before/after post-run document with an isolated evaluation. System-level results do not establish the effect of a particular evolved instruction. The trained models or ensembles produced during competitions are task outputs, distinct from changes to the agent's reusable knowledge.

DisCo: refining task-oriented research guidance. (Chen et al., 2026a, Appendix A.2.1)

Setting and seed. MLE-bench uses a fixed Codex runtime and repository-grounded guidance constructed for each competition. The initial plan scopes needed capabilities and collects relevant sources while excluding the original competition webpage and competition-specific content.

Reported evolution. Diagnostic trials produce logs and results that drive repeated revisions of modeling, preprocessing, training and checking guidance. Final skills are descriptive and exclude runnable training/inference scripts; the resulting graph is frozen for downstream execution.

Evidence and limits. Appendix A.2.1 separates construction/exploration from the matched running budget. Construction already encounters the task, so this is not unseen-task transfer. The retained object is operational knowledge around a fixed controller, not an evolved Codex runtime or a particular trained model. Reported results do not isolate the contribution of an individual ML rule.

C.5 Source Index for Coding Records

Stable IDs link the coding records to the versioned sources below. Linked names open the inspected manuscript; the following locators identify the supporting passages or artifacts. Records with inline locators need no additional ID.

S01. Multi-Task Self-Harness (v1). §3.2–3.4; §4.1 Table 3; §5.1–5.7 Table 4; Appendix C

S03. Meta-Harness (v1). §4.3; Appendix B.3 / Figure 9

S04. AHE (v4). §4.1–4.4; Appendix A Table 4; B.1

S05. AHE (v4). Appendix C.1–C.2; Figures 8–11; commits 169c34c,4e0aab9,ff0cf3d,9651986,ca35f53, a4a4a29

S06. RRSI (v2). §4.1,4.3; Appendix C; Appendix D.1 Table 5

S07. RRSI (v2). Appendix E Table 6 (Coding R0-A,R0-B,R8-B)

S08. ModularRSI (v1). §4,5.3,6.1,6.4

S09. ModularRSI (v1). Appendix D.2–D.4; Tables 10, Listings 7–9

S10. Self-Harness (v3). §3.3,4.1; Figure3; Appendix A.2

S11. Self-Harness (v3). §4.3 Figures5–8; Appendix B.1–B.2

S12. Mixture of Self-Improving Branches (v1). §4.1; Tables1–4

S13. Mixture of Self-Improving Branches (v1). Appendix E.3–E.4

S14. DarwinX (v1). §3,§8.1 Table6; Appendix B Table8

S15. RHO (v3). §5 setup; §6.1; Appendix E.1–E.2

S16. RHO (v3). Appendix H.1–H.2; §6.1

S20. HarnessBank (v2). §4.1–4.2,4.5–4.6; Table1

S22. DGM (v3). §3–4.4; Appendix C.1, E.2–E.5, F.1–F.2

S23. DGM (v3). Appendix F.1: coding_agent.py is_patch_valid/forward; tools/edit.py; llm_withtools.py summarize_messages; HTML L1747–3620

S24. SICA (v2). §3, §3.1; §4 Figure 3/Table 1; Appendix A–C

S25. TTHE (v1). §4.4–4.5; §5.1,5.6; Appendix Figure5

S26. HarnessCompass (v1). Experiments Setup; Appendix C Two kinds of accumulated memory

S29. EvolveNet (v1). §6.1–6.9; Appendix D

S34. DGM (v3). Appendix F.2: SolutionAttempt/run_tests/forward; tools/edit.py; HTML L3622–4395

S36. AIDE^2 (v1). §2.1–2.2; §3.2–3.6; Appendix A/Table 2; Appendix D–E

S52. OneRecipe (v1). §3.1; §4.3–4.6; Appendices A.6–A.8, B.1–B.3, G, H

S53. AFlow (v4). §4–5; Appendix A.3–A.6; Appendix B.1, optimal MBPP workflow (PDF p.22)

S71. SIA (v2). §3.2–3.3; §5.3; §6.1 Table 2; §6.2–6.3.3; §7.2

D Reasoning and Information Processing: Supplementary Evidence

This appendix supports Section 4.2. It groups seven subcategories into three families: mathematical reasoning (arithmetic, advanced mathematics, and formal proofs), text classification (legal and symptom tasks), and question answering (supplied records and web information seeking). The records show how task criteria shape representations, evidence selection, and corrective procedures. Each retains its own feedback and evaluation setting; tests of one shared artifact are not independent discoveries.
Figure 7 groups representative benchmarks and initial harnesses; the records below give their exact study-level correspondence.

D.1 Mathematical Reasoning

D.1.1 Arithmetic Word Problems

The comparison concerns what the harness checks before returning a numerical answer: an intermediate problem representation or a set of candidate solutions.

ADAS: checking an intermediate representation. (Hu et al., 2025, Sections 3, 4.2–4.3; Appendix G, Code 6)

Setting and seed. Search on MGSM starts from an archive containing supplied agent designs, rather than a single named seed.

Discovered behavior. The example named Verified Multimodal Agent generates a representation of the problem, passes it to a second module for feedback and revision, and conditions the final reasoning module on that revised representation. The listing passes textual fields; the name alone does not establish an image-processing pipeline.

Outcome and scope. This is a documented discovered agent, not the paper's highest-scoring named agent. Table 2 separately evaluates the top three MGSM agents on GSM8K and GSM-Hard; Appendix A adds SVAMP and ASDiv. Those transfer scores cannot be assigned to Code 6's mechanism.

AFlow: selecting and checking candidate answers. (Zhang et al., 2025b, Section 4; Figures 5–6)

Setting and seed. GSM8K search starts from an empty workflow template with supplied generation, selection and programming operators; their later call order is not part of the seed.

Discovered behavior. The selected program generates five solutions, selects with ScEnsemble, and invokes Programmer for a computational review, falling back to the selected solution if no output is produced.

Outcome and scope. Figure 6 traces accepted additions and prompt refinements. The operators are supplied; their composition is searched. The reported workflow is retained, but its aggregate score does not isolate each addition.

MermaidFlow: computational candidates and final formatting. (Zheng et al., 2025b, Appendix B.1)

Setting and artifact. The published GSM8K workflow combines one textual solution with five Programmer outputs produced under different analysis instructions, selects with ScEnsemble, then invokes Programmer to refine and format the selected answer.

Comparison and limits. Relative to AFlow's post-selection computational review, this artifact places computational generation before selection as well. The listing supports this composition, not a claim that all calls execute in parallel or that each added call improves accuracy independently.

FlowEvo: carrying useful procedures across problems. (Ren et al., 2026, Sections 3–4; Appendix D.3)

Settings and update. GSM8K and MATH-500 evaluate a workflow generator augmented with a persistent skill bank. Successful procedures are compiled and can be retrieved as context for later workflow generation.

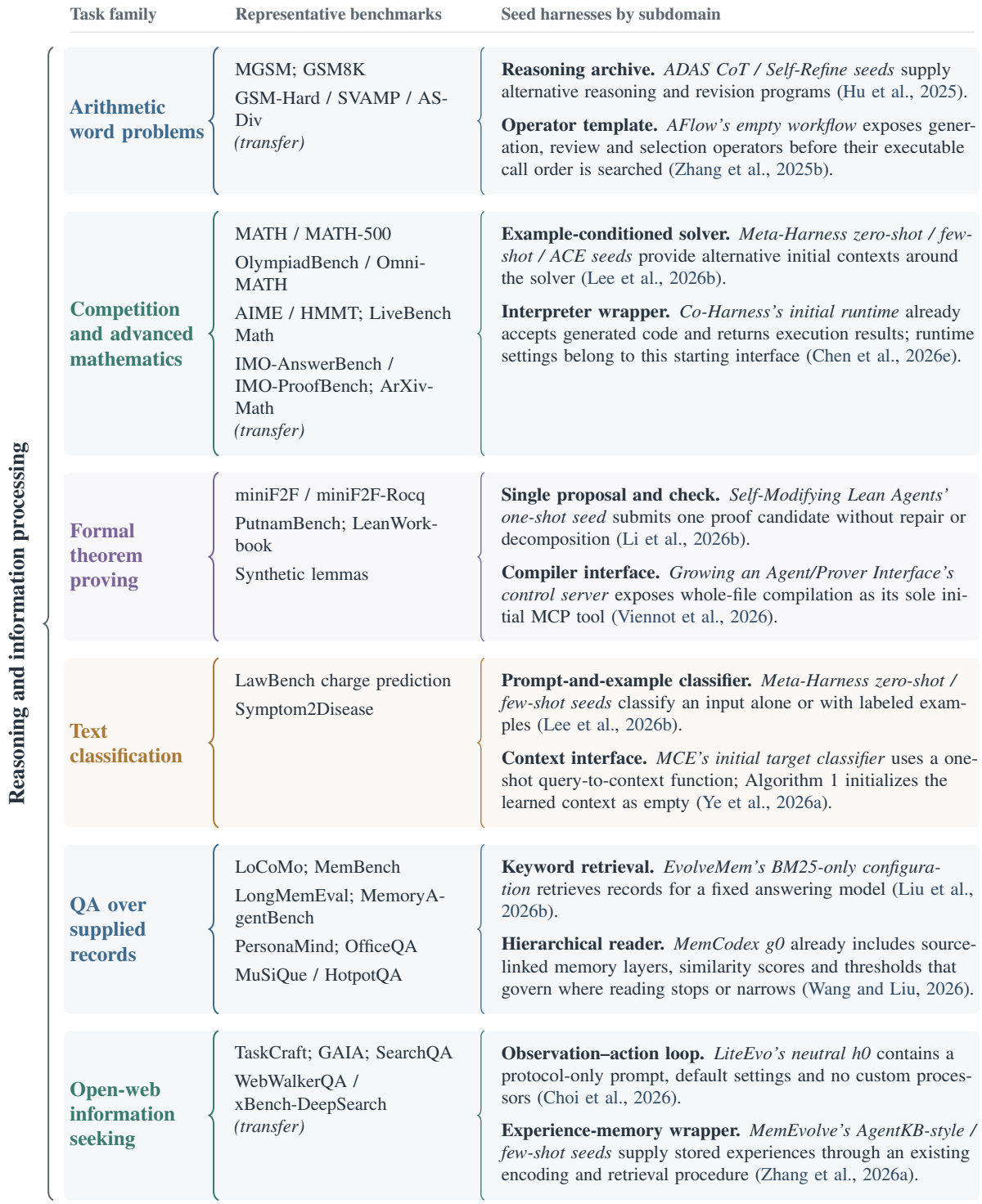


Figure 7: Representative benchmarks and seed harnesses for reasoning and information processing. Bold text describes the starting structure; italics identify an actual seed or study-specific initial configuration. Each row groups studies, without assigning every seed to every benchmark. Legal and symptom classification share an overview row but retain separate evidence subcategories. Transfer settings refer to the source lineage. Appendix D distinguishes inherited capabilities from evolved changes.

Evidence and limits. The paper identifies skill-conditioned generation as the relevant reuse mode for code and math. It does not provide the same detailed mathematical skill lineage as its ALFWorld example. We therefore record a supported accumulation mechanism and mathematical evaluation, rather than inventing a particular evolved mathematical tactic or attributing ALFWorld's replay gains to mathematics.

D.1.2 Competition and Advanced Mathematics

Here the studies change complementary parts of difficult mathematical problem solving: solution diversity, external examples, execution reliability, and reasoning control. A shared benchmark family does not make their models, task subsets, or budgets comparable.

AFlow: combining programmatic and textual solutions. (Zhang et al., 2025b, Appendix B.1, MATH workflow, PDF p. 21)
From the same template family, a separate MATH workflow executes a program, rewrites its output as a solution, generates three textual alternatives, and selects among the four with ScEnsemble. This is the published selected workflow for the level-5 setting, not a frozen transfer of its GSM8K procedure.

Meta-Harness: subject-dependent retrieval. (Lee et al., 2026b, Section 4.2; Appendix B.2; Appendix C.3)
Setting and seed. Search uses 250 OlympiadBench / hard Omni-MATH problems and zero-shot, few-shot, and ACE initializations.
Discovered behavior. The selected BM25 harness routes by mathematical subject. Combinatorics deduplicates and reranks examples; geometry combines a hard reference with lexical neighbors; number theory favors explicit techniques; the default route adjusts example count. Two search lineages contribute to the merged policy.
Outcome and scope. A frozen harness is evaluated on 200 unseen IMO-AnswerBench, IMO-ProofBench, and ArXivMath problems: average accuracy is 38.8%, versus 34.1% without retrieval. Evaluation includes the search backbone and four unseen models. These are whole-policy results, not independent ablations of routing rules or formally checked proofs.

Co-Harness: repairing execution before adding diversity. (Chen et al., 2026e, Section 4.3; Appendix H)
Setting and seed. The reported AIME24 case starts from a failing code-interpreter configuration.
Discovered behavior. Changes repair launch settings and process cleanup, batch inference, restore thinking on follow-up turns, and combine trajectories from two seeds by majority vote. Later prompt changes regress and are rolled back.
Outcome and scope. The final ensemble is reported within the time budget. We do not reproduce the claimed speedup: the stated ratio disagrees with the accompanying times. Appendix H also describes 90 questions despite the AIME24 label. These ambiguities limit quantitative interpretation, while the sequence of runtime edits is explicit. The separate AIME24/25 and HMMT25 co-evolution results also include model training.

HarnessBank: matching recovery to the backbone. (Luo et al., 2026e, Sections 4.1–4.2, 4.6; Tables 1–2)
Setting and seed. EvoAgentBench's Omni-MATH setting uses a study-specific vanilla harness and disjoint training/test tasks.
Discovered behavior. For the Qwen backbone, recovery disables thinking after a runaway rather than on every call. The evolved stack transfers to another Qwen model, but does not help Gemini's different failure pattern; its matched intervention increases reasoning instead.
Outcome and scope. Table 1 reports test success rising from 54.3% to 66.0% for the selected Qwen harness. Cross-model results support a failure-dependent intervention, not a universal setting. The full stack's gain is not attributed to the recovery switch alone.

MermaidFlow: diversified solvers followed by refinement. (Zheng et al., 2025b, Appendix B.2)

Setting and artifact. The published MATH workflow obtains five Custom responses under distinct solver/outline instructions and one Programmer result, selects with ScEnsemble, and sends the selection to a refinement call. Its reusable object is the operator graph and associated prompts.

Comparison and limits. Like AFlow, it combines textual and programmatic candidates, but the supplied composition and final refinement differ. The shared use of AFlow-derived operators makes this a related design lineage, not a fully independent invention of those operators.

Mixture of Self-Improving Branches: constructive checks and disagreement resolution. (Dong et al., 2026, Section 4.3, Figure 2; Appendix E.1–E.2)

Setting and search. Olympiad-level math uses Meta-Harness-based branch search with changing development subsets and proposal guidance. The reported Math–Gemini branches differentiate concrete answer checking and proof-obligation routing from solution construction and agreement challenges.

Concrete candidate. Appendix E.1's `gap_directed_construction` retrieves worked examples using a plan, rehearses a reference to produce a calibration note, and separates constructing an attainable answer from deriving necessary conditions. Agreement triggers a challenge; disagreement triggers a concrete check before adjudication.

Selection limit. Appendix E describes retrospectively test-best candidates from a development-ranked pool. Its 58% candidate score is not the score of the development-selected deployed head. The mechanism is documented, but those selection protocols are not interchangeable.

WDA: distributing refinement across selected stages. (Zhou and Cheng, 2026, Section 5; Appendices A.3 and A.13)

Setting and seed. LiveBench Math extends this category to challenging informal mathematical reasoning, rather than a named olympiad. Search starts from a minimal solver prompt and can revise instructions or split them across stages.

Selected structure. The published Qwen search selects two stages, [P11, P12], while the GPT search selects three, [P2, P3, P10], with their ancestries reported separately. These are fixed-per-task pipelines, not a runtime policy choosing depth for each question.

Outcome and limits. The reported selected test scores are 68.85% and 63.93%, respectively. They support selected pipeline structures, not a universal benefit from longer refinement or an isolated effect of any one stage.

WHALE: making tool use effective through joint updates. (Kim et al., 2026a, Sections 4–5; Appendices B and D.2)

Setting and seed. A study-specific tool-integrated math harness and Qwen3.5-2B are optimized using DAPO-Math-17K subsets and tested on AIME 2024/2025.

Observed contrast. The Appendix D.2 base trace never uses the interpreter and exhausts its answer budget, whereas the final model–harness pair solves the same counting problem. Harness-only search has little effect from the initial pair; alternating weight updates and harness search improves the reported results.

Attribution limit. This is evidence about a co-evolved pair and its behavior, not a final harness-only diff proving which change caused tool use. It complements fixed-backbone harness studies but cannot be pooled with them as an independent pure-harness gain.

D.1.3 Formal Theorem Proving

The correctness criterion is fixed by Lean or Rocq, while interaction procedures and persistent proof resources can change. Three research papers are distinguished from the additional benchmark-hosted RSI-Exam observations below.

Self-modifying Lean agents: bounded repair and checked names. (Li et al., 2026b, Sections 3.1, 4.3–4.5; Appendices A, C–D)

Setting and seed. A single-attempt/check seed evolves on warm-up lemmas, miniF2F-valid and PutnamBench tasks.

Discovered behavior. The accepted lineage adds bounded compiler-guided repair, batched lemma-name checks, and verified signatures. On unknown-name or type errors, the workflow can exclude a bad name, return a checked alternative and retry. The c141 branch plans and proves auxiliary lemmas before attempting assembly, but mismatched lemma contexts, assembly failures and longer-prompt risks prevent this richer structure from becoming the strongest route. The winning c144 remains mainly a whole-proof repair workflow; its proof-context labels do not establish a fully verified dependency graph.

Outcome and scope. On 244 held-out miniF2F-test problems, c144 reaches 45.1%, versus 12.7% for the seed. This comparison includes curriculum coevolution and multiple edits; it does not isolate the lemma tools or demonstrate unrestricted proof-graph improvement.

RSI-Exam: proof extraction and use of limited checks. (RSI-Exam contributors, n.d.a, Lean task contract and public rollouts; accessed 2026-10-05)

Setting and seed. The fixed-model seed proposes a proof, checks it, and repairs once; each theorem permits three model calls and four Lean checks.

Observed revisions. GPT-5.6-sol and GLM 5.3 optimizer runs fix extraction that truncated nested proofs; other revisions add compiler repairs or alter diagnostics. These are optimizer identities, not the pinned task model. In the GLM run, visible accuracy rises from 8/54 to 13/54 after extraction repair and to 20/54 after a combined diagnostic/prompt change; later variants regress.

Scope. These are version-level observations from one official resource, not independent papers. Visible checkpoint scores are not hidden-test gains. Final retention of each component is not established by the summary alone.

Evolved prover interfaces: from compilation to interactive control. (Viennot et al., 2026, Sections 2–5; Tables 1 and 8)

Seed and retained changes. Evolution starts from a Rocq MCP server exposing whole-file compilation. Retained additions include a persistent session with step/rollback/state interaction, a `try` tool that tests up to eight candidates and commits the first success, and error hints. A heavily used lemma-search proposal yields no accuracy gain and is rejected; exemplar retrieval is also rejected. File-manipulation and project-verification tools added for Phase 2 are supplied setup changes, not empirically selected mutations.

Outcome and limits. Held-out miniF2F-Rocq evaluation compares the resulting interface across four models. Its Lean port lowers cost/time per solve on a PutnamBench subset, but solves fewer problems than lean-lsp-mcp. Human setup and review remain; this is bounded interface evolution, not autonomous improvement of every system component.

ProofEvolve: verified resources that persist across problems. (Ye et al., 2026b, Sections 3 and 5.6)

Seed and change. The persistent schema library grows through extraction of kernel-checked subproofs; typed recombination exposes remaining premises as subgoals. This component-level accumulation is distinct from evolving a single target proof.

Reuse evidence. A controlled compositional study compares growing and reset libraries. A separate LeanWorkbook experiment retrieves self-generated verified proofs from disjoint source theorems, improving target success over no retrieval; random retrieval provides little benefit. That experiment supplies context for one whole-proof attempt, rather than testing typed DAG recombination. Its gain must not be attributed to every component of the full prover.

What the comparison supports. The categories expose different failure interfaces: numerical answers can benefit from candidate comparison, competition problems from better evidence and execution control,

and formal proofs from preserving checker-usable artifacts and diagnostics. These are our cross-study interpretations. More branches, longer reasoning, or richer intermediate structures are not uniformly successful; the relevant design question is which information or control change resolves the observed failure under the study's evaluation conditions.

D.2 Text Classification

D.2.1 Legal Charge Classification

Meta-Harness. (Lee et al., 2026b, Section 4.1; Appendix B.1)

Setting and seed. LawBench charge prediction. Zero-shot, few-shot, ACE and MCE initializations.

Reported evolution. The published classification frontier includes a two-call draft verifier: after an initial prediction from five similar examples, it retrieves five same-label confirmers and five different-label challengers, then asks whether to revise the draft. With fewer than five stored examples it falls back to a single few-shot call. The selected label-primed program instead combines one query-relevant example per label with nearby contrasting pairs in one prompt. These are alternative frontier programs, not successive stages of one final classifier.

Evidence and limits. These are shared classification programs evaluated across datasets, not a separately discovered legal decision procedure. Search-set selection precedes held-out evaluation.

MCE. (Ye et al., 2026a, Sections 3.1–3.4 and 4.1; Appendix D.4.2–D.4.3)

Setting and seed. LawBench. A fixed classifier receives a task prompt and the output of a query-to-context function; Algorithm 1 initializes learned context as empty. The initial pattern-extraction skill belongs to the context-engineering agent, not the classifier's execution harness.

Reported evolution. The learned target-side context directs full-case reading, extraction of discrete acts, act-to-charge matching and validation. Its retrieval program always supplies format rules and the decomposition framework, adding warnings when structural markers suggest multiple acts. The separate skill that teaches the context engineer how to construct this artifact is an optimizer-side change.

Evidence and limits. The task-agent evidence is the reusable context and query-dependent retrieval program in Appendix D.4.3. The optimizer skill in D.4.2 is a distinct object. Neither individual guidance lines nor retrieval triggers are independently ablated; the LawBench configuration differs from other studies.

Task-CoEvolve. (Miyai et al., 2026, Experimental setup; Appendix D.1)

Setting and seed. LawBench, within the three-dataset classification search. Zero-shot and all-example few-shot harnesses in the Meta-Harness implementation.

Reported evolution. The selected 20-percent-budget harness fuses word-bigram and character n-gram TF-IDF rankings, then downweights labels already represented while filling a bounded context.

Evidence and limits. One shared classifier serves Law, S2D and USPTO. The proposed benefit of word phrases for legal cases is an explanatory hypothesis, not a separately ablated legal mechanism.

DREvo. (Guo et al., 2026a, Experimental Setups; Overall Performance; Robustness Analysis)

Setting and seed. Law. No-memory and all-example few-shot configurations around GPT-OSS-120B.

Reported evolution. The proposer recalibrates historical evidence against current component state and converts it into targeted edit intent; the study evaluates the selected evolved classifier on held-out legal examples.

Evidence and limits. The reviewed primary text does not expose a legal-specific final program diff. We count the reported evolution experiment, not a new legal reasoning rule or the optimizer design as a learned task mechanism.

D.2.2 Symptom-to-Disease Classification

Meta-Harness. (Lee et al., 2026b, Section 4.1; Appendix B.1)

Setting and seed. Symptom2Disease (22 labels). The same classification seed family, evaluated on symptom descriptions.

Reported evolution. Label-conditioned retrieval and query-local contrasting examples provide reusable alternatives to placing the complete accumulated history in context.

Evidence and limits. The same frontier programs described above are reused across classification datasets; this is not a second independent discovery. Held-out classification accuracy does not establish clinical validity.

MCE. (Ye et al., 2026a, Sections 3.1–3.4 and 4.1; Appendix D.3.2–D.3.3)

Setting and seed. Symptom2Disease. The same fixed-model query-to-context interface starts without learned context in Algorithm 1. The initial symptom-profile skill guides the context engineer; it is not itself the target classifier's prompt or tool.

Reported evolution. The evolved target-side context contains distinctions between commonly confused labels. Its query-to-context program detects symptom patterns and returns relevant discriminator sections, falling back to the full guide if no condition matches. Separately, the optimizer's context-construction skill shifts from generalized profiles toward conservative evidence-backed additions and stopping criteria; those skill instructions are not executed by the target classifier.

Evidence and limits. Appendix D.3.3 exposes reusable classifier context and retrieval examples, but does not isolate their individual effects. The evidence and stopping thresholds in D.3.2 belong to the context optimizer's skill. They are neither target-runtime guards nor general diagnostic guidance.

Task-CoEvolve. (Miyai et al., 2026, Experimental setup; Appendix D.1)

Setting and seed. Symptom2Disease. Zero-shot and all-example few-shot seeds.

Reported evolution. Its selected shared classifier combines word and character retrieval and penalizes repeated-label coverage in the prompt.

Evidence and limits. S2D has a separate held-out test set, but the artifact is shared with the legal and reaction-classification settings. The tokenizer explanation is not a clinical causal result.

DREvo. (Guo et al., 2026a, Experimental Setups; Overall Performance; Robustness Analysis)

Setting and seed. Symptom2Disease. No-memory and all-example few-shot configurations around GPT-OSS-120B.

Reported evolution. Validation-guided evolution with recalibrated historical evidence produces a selected classifier evaluated on a separate S2D test split.

Evidence and limits. The experiment establishes task coverage; the reviewed text does not specify the final S2D retrieval implementation. Component-drift robustness is evidence about the optimizer, not a retained classifier feature.

HASE. (Luo et al., 2026c, Section 4.2.1; Appendices A.1, A.4 and E)

Setting and seed. Symptom2Disease. A recency-based few-shot retrieval tool and initially empty memory.

Reported evolution. Reported tool revisions add symptom-overlap or stronger lexical retrieval; across phases, tools and memory incorporate symptom extraction and label-grouped diagnostic indicators.

Evidence and limits. Training-time repair examples permit a second attempt, whereas phase-boundary review and final evaluation freeze committed files. Joint weight updates preclude interpreting the full gain as harness-only improvement.

D.3 Question Answering and Information Seeking

D.3.1 QA over Supplied Records

EvolveMem. (Liu et al., 2026b, Sections 4.1, 4.3 and 4.5; Appendix C.1)

Setting and seed. LoCoMo; MemBench. A BM25-only configuration around a fixed answering model, with semantic/structured views, name-stripped alternatives and query decomposition disabled.

Reported evolution. Failure-log diagnosis activates name-stripped alternative retrieval and multi-hop query decomposition, then introduces a verification pass for low-confidence answers. The first two are described as dormant configuration capabilities rather than newly authored tools. Guarded updates can revert regressions; the evolved object is the retrieval/answer configuration.

Evidence and limits. Section 4.5 removes components and reruns evolution, providing procedure-level ablation evidence rather than a fixed-checkpoint edit test. Per-question failure logs provide supervision, so this is not label-free adaptation. Table 4 and Appendix C.1 present different round-specific configurations whose alignment is unverified here; we do not merge them into one quantitative trajectory.

MemCodex. (Wang and Liu, 2026, Sections 2, 4.2–4.3; Figure 5)

Setting and seed. LoCoMo; LongMemEval; MemoryAgentBench; PersonaMind. Initial hierarchical memory architecture g0 with executable layer methods.

Reported evolution. Evolution rewrites construction, indexing, retrieval and routing programs within the supplied hierarchy. Its initial architecture already supports threshold-based adaptive reading; the evolved artifact changes those methods rather than introducing adaptive depth from scratch.

Evidence and limits. Initial hierarchy design is not itself discovered. The study separately analyzes question-level and record-disjoint splits, acceptance gates and cross-benchmark transfer; no universal optimal depth follows.

HarnessEvolve. (Jiang et al., 2026b, Section 4.1; Tables 2–3; Appendix Figure 3(a))

Setting and seed. OfficeQA. OpenClaw with initial skills, AGENTS.md, SOUL.md and tools.

Reported evolution. Figure 3(a) expands an existing financial-metrics skill: fix the statistic and operands before computing, distinguish changes in shares from growth rates, specify the denominator, convert log returns when the requested quantity requires it, and check constants and revised values against the corpus. These are reusable instructions, not enforced executable checks. The old skill already requests corpus-grounded currency conversion; that inherited instruction is not a new discovery. Selected skills are subsequently tested in other execution frameworks.

Evidence and limits. Appendix A, Figure 3(a) supplies the inspected before/after artifact; the individual rules are not separately ablated. Framework transfer reuses learned skills rather than adding independent discoveries, and the illustration alone does not establish each line's presence in the best final checkpoint.

HCL. (Kang et al., 2026, Sections 4.1 and 4.3.1; Appendix C.1)

Setting and seed. MuSiQue and HotpotQA within a textual task stream. A frozen model with editable task interface, memory, capabilities and routing.

Reported evolution. Sequential harness updates are tested on current tasks and historical anchors; the study contrasts strict retention with permissive adaptation.

Evidence and limits. Separate adaptation, validation and test examples support a continual-evolution experiment, but no final QA-specific executable diff is exposed in this record.

D.3.2 Open-Web Information Seeking

MemEvolve. (Zhang et al., 2026a, Section 4; Section 9.1; Figures 8–10)

Setting and seed. TaskCraft; GAIA; WebWalkerQA; xBench-DeepSearch. AgentKB or a simple few-shot experience memory.

Reported evolution. Search changes memory encoding, retrieval and management programs. From a lightweight AgentKB-style seed without its offline knowledge base, Cerebra adds reusable tool and knowledge distillation and working-memory maintenance; other reported branches include Riva and Lightweight.

Evidence and limits. WebWalkerQA and xBench use architectures evolved on TaskCraft. Their evaluations are frozen transfer evidence, not independent in-benchmark evolution runs.

HarnessX. (Chen et al., 2026d, Experimental setup; Appendix 12.1)

Setting and seed. GAIA. HarnessX task configuration with prompts, processors, slots and tool resources.

Reported evolution. The GAIA evolution analysis reports shipped prompt, processor, configuration and tool edits, addressing blocked sources alongside reasoning and parsing failures.

Evidence and limits. Lever-level statistics do not identify one universally retained retrieval tool. The same task pool is evaluated during search; this supports benchmark adaptation rather than a sealed generalization test.

LiteEvo. (Choi et al., 2026, Sections 3.1–4; Appendix C.4)

Setting and seed. GAIA. A neutral observation–action prompt, default runner settings and no custom processors.

Reported evolution. Trajectory analysis proposes prompt fragments, knowledge, processors and settings; replay removes inert or invalid processors before the composer assembles candidates.

Evidence and limits. GAIA is evaluated transductively. Appendix C.4 documents answer-copy exposure despite blocking known mirrors, including a returned processor's guidance about dataset mirrors. The score does not establish clean unseen-question generalization.

HarnessEvolve. (Jiang et al., 2026b, Sections 3–4; Tables 2–3)

Setting and seed. SearchQA. OpenClaw with an unoptimized skill and tool configuration.

Reported evolution. Failed executions and validated answer-assisted reference trajectories drive persistent skill-directory and harness updates. The resulting SearchQA skills are frozen for cross-framework evaluation.

Evidence and limits. The SearchQA scores and transfer establish task-level evolution evidence, but the OfficeQA or enterprise SQL example must not be relabeled as a SearchQA-specific final edit.

E Interactive and Embodied Tasks: Supplementary Evidence

This appendix supports Section 4.3. Browser interaction, applications and services, game playing, and robot manipulation form four subcategories. The records compare how persistent changes bind actions to interfaces, encode action prerequisites, and recover from state-dependent failures. They distinguish evolved skills or instructions from supplied retry loops and frozen actors. Figure 8 summarizes the starting structures; study entries retain the specific transfer and evaluation conditions.

E.1 Browser-Mediated Interaction

SkillWeaver. (Zheng et al., 2025a, Sections 2.2–2.3; Appendix D.1.2–D.2)

Setting and seed. WebArena websites and held-out web tasks. A web agent using low-level browser actions.

Reported evolution. Practice discovers and refines executable Playwright APIs. In the published `delete_merged_branches` case, a global textbox selector matches both GitLab search and the confirmation field, causing a strict-mode error. The repaired API scopes the textbox and confirmation button to the deletion dialog; the agent patches its persistent knowledge base.

Evidence and limits. Functions persist beyond their discovery tasks. This is component-level skill improvement, not self-rewriting of the outer optimizer; dynamic-site skills can fail when realistic test inputs are unavailable.

Evo-Harness. (Wei et al., 2026b, Sections 3–4; Appendix A; Table 6)

Setting and seed. WebArena-Infinity. Solver without an evolved skill library.

Reported evolution. Online reflection is compiled into reusable general and task-type skills. Published web cases include disabling an existing auto label, preserving corporate email domains, and archiving qualifying read-important emails.

Evidence and limits. The main sequential protocol learns between batches. Frozen transfer is evaluated separately; the web examples are advisory procedures, not verified new browser APIs.

LiteEvo. (Choi et al., 2026, Section 3.1; Appendix B.1–B.2; benchmark-specific analysis)

Setting and seed. WebShop. A neutral turn-protocol prompt and action–observation chat loop, with no initial processors or benchmark-specific prompt content.

Reported evolution. Repeated failure analysis is converted into reusable prompt, knowledge, processor and configuration components, with candidate selection under the WebShop task evaluator.

Evidence and limits. The study reports an inductive WebShop configuration, but the concrete login processor described for AppWorld is not a WebShop discovery. A complete selected WebShop diff is not established here.

HarnessX. (Chen et al., 2026d, Experimental setup; Appendix Additional Results, WebShop)

Setting and seed. WebShop. Composable HarnessX web-task configuration.

Reported evolution. Reported prompt/configuration edits reduce search and pagination loops; advisory processors address navigation. The added color-matching tool has zero measured hit-rate in the reported run.

Evidence and limits. Search repeatedly evaluates the full selected task pool. Aggregate improvement and edit counts do not establish unseen-site transfer or the causal effect of one component.

E.2 Application and Service Workflows

Task-time service-control reference. Difficulty-routed service control combines a lightweight routing decision with conflict-aware candidate generation and verification before state-changing backend writes,



Figure 8: Representative starting harnesses for browser, application, game and robot tasks. Bold labels describe the initial interface, and italic examples identify its study and configuration. Supplied tools, retry loops and frozen actors are not themselves evolved discoveries. Rows group settings rather than establish all-to-all benchmark–seed matches. Appendix E records the specific changes, retained artifacts and transfer conditions.

evaluated on retail and airline tasks from τ^2-bench (Chen et al., 2026b, Section 3). This provides a concrete design reference for selectively allocating deliberation and checking action targets, scope, and ordering. The architecture is supplied rather than discovered through cross-round harness evolution; it is therefore background for the evolution records below, not an additional evolved-harness finding.

LiteEvo. (Choi et al., 2026, Sections 3.1–3.3; Appendix B.1–B.2; AppWorld evaluation)

Setting and seed. AppWorld. The same neutral turn protocol with a code-action format; no initial processors or learned task components are supplied to the worker.

Reported evolution. The returned harness adds a worked login example and credential-field knowledge; a processor restates the login steps after two authorization errors. These components augment the initially neutral code-action loop.

Evidence and limits. The processor is a reported concrete returned component. Table 2 gives transductive AppWorld pass@2 of 0% with processors alone, 63.1% with the prompt and 67.7% for the full harness; this separates component bundles, not the login processor individually. The inductive protocol separately chooses on validation tasks and tests unseen tasks. Transfer of components evolved on other benchmarks is much weaker and must remain distinct.

ScholarEvolve. (Yang et al., 2026b, Section 4; Appendix A; Appendix D.2, Table 8 and final-module listings)

Setting and seed. AppWorld; τ^2-Bench Telecom. Initial modular application/service agent.

Reported evolution. Validation selects combinations of research-derived tool-interface, context, skill, memory and reasoning modules. Table 8 identifies different selected modules for each backbone and benchmark; final AppWorld excerpts show their implementations.

Evidence and limits. Paper-derived module names identify design provenance, not proof that the complete source method was reproduced. Periodic lifelong refresh is distinct from the main finite module-search experiment.

Evo-Harness. (Wei et al., 2026b, Sections 3–4; benchmark and skill analyses)

Setting and seed. τ-bench. Solver without an evolved skill library.

Reported evolution. Batch-level feedback supports addition, merging and revision of reusable service-task guidance, evaluated in the paper's online harness-learning protocol.

Evidence and limits. The service experiment qualifies as persistent skill evolution, but the detailed web examples above cannot be assumed to be its retained service policies.

HarnessX. (Chen et al., 2026d, Experimental setup; Appendix Additional Results, τ^3-Bench)

Setting and seed. τ^3-Bench. HarnessX dialogue and tool-use configuration.

Reported evolution. The reported service runs ship prompt rules and processors addressing policy ordering and premature actions. No tool edits are shipped in these runs.

Evidence and limits. This is a distinct experiment from WebShop, not an additional paper. Repeated search-pool evaluation and separate co-training experiments must remain distinct from frozen-model generalization.

RRSI. (Xia et al., 2026, Section 4.1; Appendix A; Appendix E)

Setting and seed. Harvey LAB; transfer to JobBench, GDPval and APEX-Agents. A study-specific ReAct loop over an MCP gateway with a dynamic toolbelt and ReSum-style context management.

Reported evolution. Regularized edits optimize the workspace harness on Harvey LAB, and the resulting harness is evaluated on a pristine in-domain split and three unseen professional-work suites.

Evidence and limits. The paper's detailed accepted-edit table contains coding and engineering examples, not a Harvey-specific diff. We record workspace evolution and transfer without importing those other domains' mechanisms.

E.3 Game Playing

AutoHarness. (Lou et al., 2026, Sections 3–4; Appendices C–D)

Setting and seed. TextArena. A fixed LLM-action/retry scaffold with function interfaces for synthesizing action proposals and legality checks. The LLM without this harness is a comparison baseline.

Reported evolution. Environment feedback fills and refines the supplied `propose_action` and `is_legal_action` interfaces. In the verifier configuration, the existing loop checks LLM-proposed moves and returns an illegal-action warning before retrying. Appendix D.2 exposes chess coordinate parsing, king location and attack detection; a separate configuration synthesizes whole Python policies.

Evidence and limits. We count the verifier configuration as a harness around an LLM. Pure code policies replace task-time LLM execution and are a distinct boundary case; neither changes the outer search rule.

Continual Harness. (Karten et al., 2026, Sections 2–4; Appendix C)

Setting and seed. Pokémon Red and Emerald. A minimal screenshot/map/button interface with no curated skills or subagents.

Reported evolution. An automated Refiner revises prompts, subagents, executable skills and memory during a continuous run, using navigation loops and tool failures as feedback.

Evidence and limits. This record concerns the automated configuration. Historical Gemini Plays Pokémon milestones involve human refinement; detailed historical battle-agent changes cannot be attributed to the automated experiment. Model co-training is separate.

DemoEvolve. (Che et al., 2026, Sections 4.1–4.3; Appendices B, C and G)

Setting and seed. Liar's Dice; Balatro; Slay the Spire 2. Game-specific base harnesses with fixed environment interfaces.

Reported evolution. Selected Liar's Dice self-rollout harnesses parse game state, calculate claim probabilities and legal raises, and compare calling with raising. Appendix B gives simplified sketches of selected harnesses. Its OneCall-Wild1 variant uses a Bayesian escalation model; during deep same-face escalation, with at least two own contributions, it overrides raising when calling's expected value is higher by a margin. On Balatro and Slay the Spire 2, successful demonstrations supply a separate source of evidence for persistent harness revision.

Evidence and limits. Appendix G audits a selected Meta-Harness baseline whose proposed Balatro hook never reached the model. That inactive hook is negative evidence, not a successful DemoEvolve discovery; held-out game seeds are evaluated separately.

HCL. (Kang et al., 2026, Section 4.2.2; Appendix C.3)

Setting and seed. Minecraft 50-task curriculum. Static task interface and capability library around a frozen model.

Reported evolution. Experience supports capability and workflow revisions; retained skill tests gate later updates, extending reuse across crafting and dependent operations.

Evidence and limits. This is one sequential Minecraft experiment. It documents retention-based capability evolution, without establishing that its updater itself evolves or exposing every final skill diff.

E.4 Robot Manipulation

Zetta. (Ding et al., 2026, Sections 3–4; Appendix C)

Setting and seed. LIBERO-Pro; RoboCasa. Frozen VLA policy with an initially unevolved critic/recovery harness.

Reported evolution. The stove-placement lineage adds object-relative pregrasp. After grasp failure it releases and restages, chooses a proposal different from remembered failed grasps, and checks stability

within a retry budget. Placement lowers the object until support is detected, then releases and retreats until separation is confirmed.

Evidence and limits. Code critics propose interventions; the fixed orchestrator authorizes them and the official task predicate determines success. Section 4.3 applies the cumulative stove-task checkpoints to PnP-Sink, PnP-Cabinet and PnP-Toaster without further evolution: mean success rises from 64% to 84%. This supports reuse of the complete recovery stack, not each branch independently. Historical regression and held-out checks gate reusable updates.

SHAPER. (Wang et al., 2026a, Section 4.2; Appendix A.1, Figures A1–A2)

Setting and seed. VLABench. Generic planner guidance and a context builder accumulating all prior rounds.

Reported evolution. The evolved skill emits actor-compatible primitive commands with exact entity names. The final context builder emphasizes current images and recent execution state, compresses older rounds and exposes stagnation.

Evidence and limits. VLA weights and low-level action generation remain fixed. The before/after artifacts support actor-aware command and context changes; ESI-Bench active-perception changes are a separate setting.

HarnessPAI. (Wang et al., 2026c, Sections 4.4–5.4; Appendix C)

Setting and seed. LIBERO/Pro; RoboCasa; other embodied suites. Code-as-policy interface around fixed action primitives.

Reported evolution. Execution failures revise programs and become node-local skills. The retained `align-object-to-basket-mouth` skill diagnoses a 5.3 cm depth error caused by estimating the target from all basket-mask points. It changes `basket_center_world` to use medians of upper-rim points for both horizontal position and height, with an all-points fallback when fewer than five rim points remain. The existing placement servo is unchanged.

Evidence and limits. The program stays fixed within each rollout and evolves between rollouts. Post-training on collected expert trajectories is a separate weight-update experiment; the published skill is not an isolated causal ablation.

Recursive Harness Distillation. (Kim et al., 2026b, Sections 3.4, 4.3 and 5; Figures 6–7)

Setting and seed. SimplerEnv Bridge uses frozen GR00T; three real-world tasks use a frozen fine-tuned $\pi_{0.5}$ policy. A light agent has an observation/intervention interface, and playbook refinement starts from teacher guidance distilled from initial development rollouts; no-playbook agents are comparison baselines.

Reported evolution. Strong-agent experience and recipient feedback revise a reusable intervention playbook: re-identify occluded objects, distinguish weak from empty grasps, inspect action mismatches, and verify support before release.

Evidence and limits. The text guides interventions through the existing policy. On the same 48 held-out Bridge instances, Luna succeeds at 43.8% without a playbook, 31.3% with the initial teacher-distilled playbook and 66.7% with the refined playbook. A Luna-derived refined playbook instead reaches 22.9%; stronger-teacher guidance is not interchangeable with self-refinement. These whole-playbook comparisons do not isolate individual rules or show that the distillation algorithm rewrites itself.

F Visual Understanding and Generation: Supplementary Evidence

This appendix supports Section 4.4. Visual understanding covers video QA, temporal grounding, and image perception; visual generation covers image generation and video production. The nine category–study records from nine papers compare observation and evidence handling with reusable composition and selection procedures. Figure 9 identifies representative starting structures. Each record states whether the source exposes a concrete changed artifact, an illustrative intermediate variant, or only the update procedure.

F.1 Visual Understanding

F.1.1 Video Question Answering

Video-RSI. (Luo et al., 2026b, Sections 4.1, 4.3–4.5; Table 2)

Setting and seed. LVBench evolution; frozen evaluation on Video-MME, LongVideoBench, EgoSchema and MLVU. The seed already has global/local inspection, transcript/OCR access and observation memory.

Reported evolution. Retained revisions add frame-linked findings with uncertainty, relations over existing observations, selective 64-frame overviews and occurrence counting. Graph queries process no new frames; counting consolidates overlapping detections.

Evidence and limits. The full harness uses fewer frames and answers more accurately than the seed on the four reported suites. Table 2 ablates editor investigation and selection gates, not individual deployed tools. Frame savings exclude editor cost and are not total-compute savings.

VidHarness. (Liang et al., 2026, Sections 3–4; Appendix D)

Setting and seed. LongVideoBench; Video-MME; Video-Holmes; transfer to Video-MMMU/MMVU. A task contract and video/transcript/VLM APIs for constructing executable video-agent programs; no separately named seed is established here.

Reported evolution. MCTS with multi-fidelity validation searches selective-observation programs; retained harnesses form budget-specialized choices in a mixture.

Evidence and limits. Search design and budget routing are specified by the method. The reviewed text documents evolution and validation trajectories more clearly than final per-tool diffs, so we do not invent a discovered frame-sampling rule.

F.1.2 Temporal Grounding

VideoEvolve. (Luo et al., 2026a, Sections 4.1, 4.4–4.5; Table 3; Figure 3)

Setting and seed. Charades-STA; ActivityNet; QVHighlights; ReXTime; TimeLens-Bench. The seed predicts one interval from 64 uniform frames and generic instructions.

Reported evolution. Instructions check repeated occurrences and event boundaries. Coarse sampling or unusual interval length triggers another prediction; substantial disagreement triggers a third. Start and end times are separately combined by medians.

Evidence and limits. All compared configurations use the same 64 frames; extra calls add computation. Figure 3 shows a recovered kettle interval. For the light switch, the instruction-only variant already moves the correct seed interval $[0, 1]$ to $[1, 3]$; the three-call variant returns $[1, 4]$. Extra calls therefore do not recover this instruction-level regression. These are cases; Table 3 separates code and instruction contributions across settings.

F.1.3 Image Perception and Question Answering

HCL. (Kang et al., 2026, Section 4.3.2; Appendix B.2, Table 10; Appendix C.2)

Setting and seed. COCO detection/captioning; RefCOCO; VQAv2. Frozen VLM with editable external task interfaces, memory, capabilities and router.



Figure 9: Visual task families and their starting harnesses. Named examples describe supplied observation, localization or generation procedures, including any initial checking or memory. They do not describe the later evolved mechanisms. VidHarness provides a task/API contract, but the reviewed source does not establish a distinct initial sampling program. Benchmarks include direct evolution and transfer settings; exact assignments and retention limits appear in Appendix F.

Reported evolution. Sequential non-parametric updates balance learning the next visual task against preserving previously solved anchor examples.

Evidence and limits. Disabling memory updates mainly reduces captioning performance; fixing the router produces its largest task-level decline on VQAv2. Interface updates contribute most visibly to captioning and VQAv2. These ablations support different component roles, not a published final grounding-specific rule; variants follow different update trajectories. This is the same paper as the QA and Minecraft records.

F.2 Visual Generation

F.2.1 Image Generation

AutoRef. (Oshima et al., 2026, Sections 4–5 and 6.5; Table 4; Appendices D and H.2)

Setting and seed. MultiBanana; OmniContext. Search starts from a one-call generator and GEMS.

Reported evolution. The selected lineage binds prompt elements to references, generates subject-led and background-led drafts, checks hard failures before order-swapped comparisons, and generates a third draft from reference-specific complaints. When hard-failure counts tie, the challenger replaces the incumbent only if it wins both presentation orders. This selector uses the target reasoning model; the search evaluator is separate.

Evidence and limits. Direct-edit variants leave the retained lineage. The fixed-three-image ablations replace removed drafts with another sample and replace failure-aware selection with Best-of-N. They support the combined selector rather than isolating order swapping. Harness-search cost is separate; generator weights stay fixed.

COMFYCLAW. (Li et al., 2026d, Sections 3.4–4; Table 2; Appendix C, Table 8)

Setting and seed. GenEval 2; DPG-Bench; OneIG splits. A ComfyUI graph-construction agent starts with predefined tools and four general image-generation skills. The supplied refinement architecture includes a visual verifier; the Base comparison executes once without verifier feedback.

Reported evolution. Cross-run traces and feedback produce reusable `SKILL.md` procedures; reported examples include spatial/count binding, attribute binding and benchmark-dependent style or detail guidance.

Evidence and limits. Typed editing, invalid-edit rollback and the region verifier are supplied architecture. Skill-read counts establish consultation, not isolated causal quality. The reported workflow-edit distribution describes per-image actions; it does not by itself identify persistent edits to the harness.

EvoGen-Harness. (Luo et al., 2026d, Sections 3–4; Appendices B.6–B.7)

Setting and seed. GenEval2; T2I-CompBench++; WISE. Frozen image generators wrapped by an initial multi-responsibility harness.

Reported evolution. Figure 6 illustrates localized counting-policy, spatial-composition, attribute-binding tool-knowledge and action–object memory changes.

Evidence and limits. Earlier Appendix B examples phrased as possible patches illustrate the edit space, rather than establish exact retained diffs. Trace, No-Patch and held-out gates are supplied evolution machinery. Figure 6 gives qualitative cases; aggregate scores do not isolate each patch. The experiments adapt a separate harness for each generator, rather than transfer one unchanged harness across them.

F.2.2 Video Production

VideoWeaver. (Wei et al., 2026a, Section 4.2; Sections 5.2–5.3; Appendix C)

Setting and seed. Long-video production categories and held-out cases. Fixed foundation skills plus initial category composition and creator skills.

Reported evolution. Training feedback revises reusable composition skills and the instructions that create them; category-local creator skills are merged for unseen-category transfer.

Evidence and limits. Composition skills are frozen for held-out cases; the merged creator is frozen before receiving unseen-category skill queries and does not update on OOD execution feedback. Section 4.2 and Appendix C specify the update and merging procedures, but the reviewed text does not establish an exact final per-category instruction diff.

CineForge. (Liu et al., 2026c, Sections 3.3 and 5.5; Appendices A.6, B.1.6 and B.3–B.8)

Setting and seed. CineScope; AnimeShooter; ViStoryBench. CineForge-Produce starts with a fixed initial production policy and supplied production stages.

Reported evolution. Cross-story evidence produces bounded stage-local changes to a persistent textual production policy, admitted through fresh replay and, where needed, paired stochastic evaluation.

Evidence and limits. The permitted patch types in Appendix B.3 are not a list of exact retained edits. Three-round checkpoint evaluation is read-only on a fixed unseen split. Review-workload measurement overrides editable operational-review settings with a frozen audit reviewer, so lower workload does not result from simply weakening that reviewer.

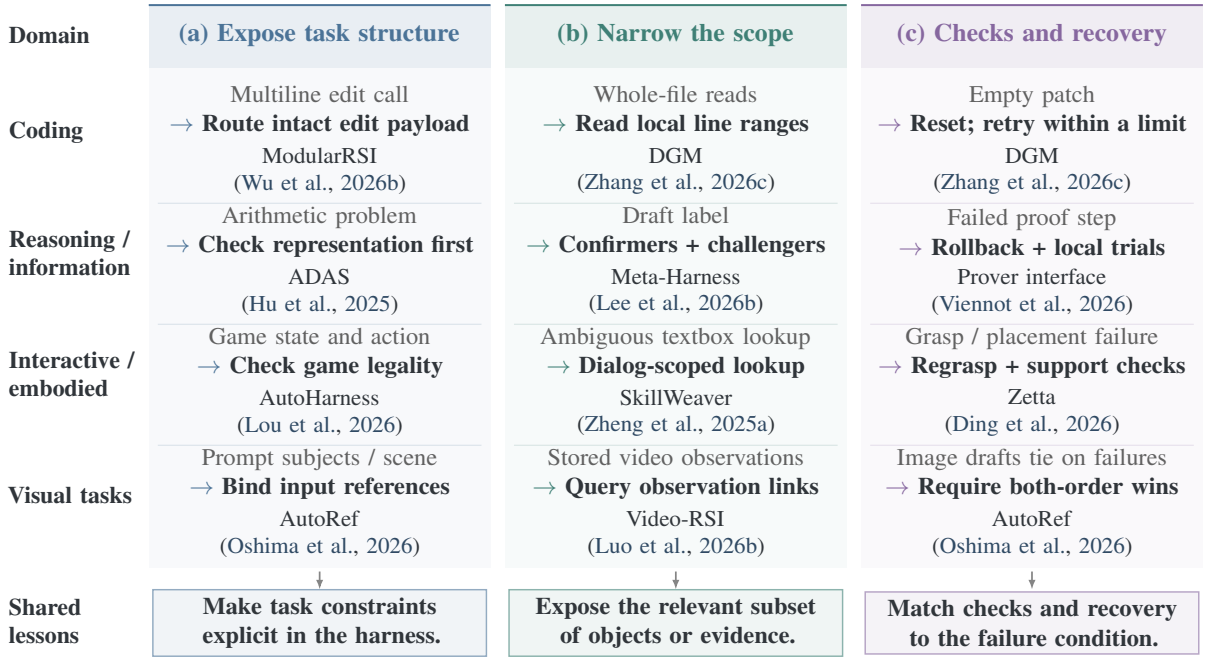


Figure 10: What target harnesses change across domains. Each cell links a task input or condition to reported reusable handling; the bottom row gives our cross-study synthesis. Entries compare distinct artifacts, not a joint pipeline or proven transfer. ModularRSI shows a historical revision, Meta-Harness a frontier variant, and AutoHarness evolved legality code within a supplied loop. Appendices C–F give artifact status, outcomes, and limits.

G Cross-Domain Comparisons and Evaluation

This appendix supports Section 4.5; it is not an additional task domain. The mechanism matrix complements the evidence-set overview in Figure 3. The following two groups compare optimizer benchmarks and task-specific contracts, asking what seed is supplied, which feedback search can access, and what the final evaluator tests. These six protocol records are separate from the 95 domain records in Appendices C–F. Official task pages were checked on 6 October 2026; their protocols and displayed runs may change.

G.1 Concrete Mechanisms across Domains

Figure 10 aligns task conditions with reported revisions. Shared functions motivate comparison; their implementations and verification conditions differ. The matrix indexes selected cases from Appendices C–F, rather than adding independent records or establishing cross-domain transfer.

G.2 Controlled Comparison of Optimizers

HarnessOpt-Bench. (Ursekar et al., 2026, Sections 3–4; Appendices A–B)

Setting and seed. OfficeQA, BrowseComp-Plus, Terminal-Bench and GAIA use pinned executable seeds. Three are deliberately basic working agents; GAIA starts from a non-functional stub.

Optimization contract. Code changes run against a fixed target model, environment and verifier. Development feedback includes case outcomes and traces; validation exposes aggregate scores. Search nominates a final candidate before trusted test evaluation.

Evidence and limits. Search cannot access the final test. Evaluation-call, case-pass and target-token allowances constrain target evaluation; optimizer inference is recorded but uncapped. GAIA gains include construction, so seed strength and search expenditure must accompany comparisons.

Cross-domain optimization benchmarks

Setting	Representative benchmarks	Seed harnesses by setting
Task-specific optimizer comparison	HarnessOpt-Bench OfficeQA; BrowseComp-Plus Terminal-Bench; GAIA	**Three working seeds and one construction stub.** *Pinned task agents (HarnessOpt-Bench).* Basic Python tool-use agents serve the first three tasks; the GAIA seed is non-functional (Ursekar et al., 2026).
Shared-harness comparison	Evo-Bench BrowseComp; HLE; GDPval APEX-Agents; Claw-Eval	**One shared executable loop.** *Minimal CodeAct (Evo-Bench).* Shell execution and final-answer submission form the common seed; the fixed outer evolution harness is separate (Huang et al., 2026a).
Proof-search contract	RSI-Exam Lean workflow	**Bounded proposal and repair.** *One-proposal/one-repair Lean seed.* The program drafts a proof, invokes the compiler and permits one diagnostic-guided repair (RSI-Exam contributors, n.d.a).
Memory contract	RSI-Exam Conversational memory	**Extraction, retrieval and answering.** *BM25 baseline.* The supplied program extracts conversational information, builds a keyword index and returns concise answers (RSI-Exam contributors, n.d.b).
Discovery contract	RSI-Exam DiscoveryWorld	**History-carrying action loop.** *ReAct starter.* Stateless model calls receive action–observation history (RSI-Exam contributors, n.d.c).
Swarm optimization contract	Scale RSI Bench Agent-swarm optimization	**Graph with trainable edge probabilities.** *GPTSwarm-style REINFORCE seed.* A supplied policy-gradient optimizer learns connections among fixed-model nodes (Scale AI, n.d.).

Figure 11: Starting systems in cross-domain harness-optimization resources. Each row identifies a resource and its supplied seed; these are evaluation settings rather than additional task domains. HarnessOpt-Bench includes both working agents and a construction stub, whereas Evo-Bench uses one shared CodeAct seed. Editable components alone do not establish evolved functionality or recursive reuse of an optimizer. Appendix G gives feedback, budget and final-evaluation boundaries.

Evo-Bench. (Huang et al., 2026a, Section 3; Section 5.1)

Setting and seed. One shared CodeAct loop, initially exposing shell execution and final-answer submission, serves search, office and general-agent tasks with a fixed policy model.

Optimization contract. Evolution can use 160 visible cases drawn from BrowseComp, HLE, GDPval, APEX-Agents and Claw-Eval.

Evidence and limits. The final revision is frozen for 448 disjoint cases, testing one shared harness across task families. It need not be the highest-scoring visible checkpoint.

G.3 Task-Specific Optimization Contracts

RSI-Exam: Lean proof-search workflow. (RSI-Exam contributors, n.d.a, Official task: What You Have; How It Is Judged)

Setting and seed. The supplied program proposes a proof, invokes Lean, and allows one diagnostic-guided repair; 54 visible theorems support development.

Optimization contract. The solver may revise proof extraction, proposal and repair control under fixed model and execution budgets.

Evidence and limits. Hidden evaluation compiles proof bodies against canonical statements. The contract establishes no retained improvement by itself; the formal-proof record describes a reported trajectory.

RSI-Exam: long-term memory architecture. (RSI-Exam contributors, n.d.b, Official task: What You Have; How It Is Judged)

Setting and seed. The baseline extracts conversational information, indexes it with BM25 and produces concise answers.

Optimization contract. Memory construction and question answering are editable behind fixed ingestion and answer interfaces, with a pinned model.

Evidence and limits. The sealed verifier reruns the full pipeline on unseen conversations and scores token F1 within a wall-clock limit. Visible-set changes and scores do not by themselves establish held-out improvement.

RSI-Exam: scientific-discovery agent. (RSI-Exam contributors, n.d.c, Official task: What You Have; How It Is Judged)

Setting and seed. A ReAct starter sends action–observation history to a stateless model call at each turn. Six visible DiscoveryWorld instances support search.

Optimization contract. The editable program controls memory, observation selection and actions through the supplied environment interface.

Evidence and limits. Evaluation uses unseen parameterizations of the same two themes, a fixed model and a 240-action budget. Progress, completion and explanatory knowledge contribute separately; this is not transfer to unseen scientific themes.

Scale RSI Bench: agent-swarm optimization. (Scale AI, n.d., Official task: Task Description; What Did We Learn)

Setting and seed. A REINFORCE optimizer learns graph-edge probabilities around a fixed node model.

Reported evolution. Official runs replace policy-gradient search with direct graph search, cached rollouts and replay; retained graphs combine solving, code verification and final decisions.

Evidence and limits. Hidden AIME evaluation scores a frozen graph on accuracy, call efficiency and validity gates. Optimizer redesign alone does not establish recursive reuse.

G.4 Adjacent Mechanisms and Post-Cutoff Pointers

These references clarify the boundary between reusable harness mechanisms, learned policies, and task-level solution search. They supplement the discussion without changing the frozen evidence records or benchmark counts.

Diagnostic tools and learned use. MLToolBench constructs diagnostic tools from recurring ML-agent failures, re-tests candidates on the original tasks, and retains useful tools in a reusable library (Yu et al., 2026b, Section 2.1). Its subsequent supervised and reinforcement learning train the policy to use that interface. Tool-library construction and policy training are distinct; aggregate training gains are not evidence that the diagnostic interface evolved during evaluation.

Reusable preference context. EXACT selects interpretable attribute subsets from pairwise preference feedback, stores them in a per-user prompt–attribute index, and retrieves relevant attributes to augment new prompts with base-model weights fixed (Yu et al., 2026a, Sections 4.1–4.2). The reusable object is a preference-context index over a predefined attribute vocabulary, not a newly discovered controller or evidence that the optimization procedure recursively rewrites itself.

Verification-efficient improvement: post-cutoff pointer. First released on 6 October 2026, after the 1 October evidence cutoff, *Verify Less, Evolve More* trains idea-level critics to screen proposed ML changes before costly execution, integrates screening into OpenEvolve, and separately studies critic adaptation and proposer-policy training (Bai et al., 2026, Sections 2, 4–5). This is relevant to verification allocation and learned improvement components; evolving task programs is not by itself evidence of reusable target-harness evolution. It is cited here as a dated follow-up, not added retrospectively to the frozen comparison.

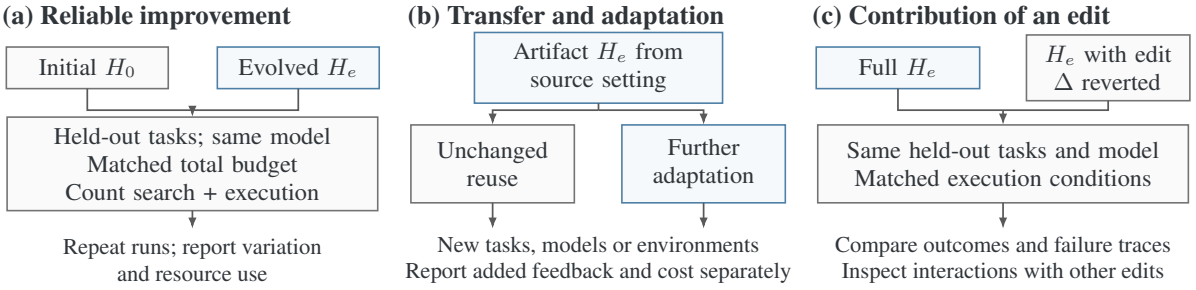


Figure 12: Conceptual evaluation comparisons; H_e is a selected evolved harness. Reliability, transfer, and edit attribution require different controls; track regressions and safety violations. Adaptation feedback must remain separate from final test evidence. A changed improver additionally requires a comparison of subsequent improvement, as discussed in Section 3.2.

H Challenges and Future Directions

This appendix follows the three priorities in Section 5. Appendix H.1 combines transfer conditions with reliable, cost-aware measurement. Appendix H.2 examines attribution, edit scope, and coupled model updates. Appendix H.3 addresses constraints across successive revisions. Each topic connects reported mechanisms and results to the remaining comparison problem; numerical results retain their original evaluation settings. Appendix A.3 describes the corresponding workflow designs.

H.1 Measure What Transfers

H.1.1 Generalization beyond the Evolution Setting

Across tasks and benchmarks. Repeated evaluation on a finite search set can favor task-specific fitting, noise chasing, and unnecessary complexity. RRSI addresses these risks through regularized proposals and acceptance (Xia et al., 2026). In its workspace setting, the harness evolved on Harvey LAB raises the mean score on JobBench, GDPval, and APEX-Agents from 39.7 to 43.6. HarnessCompass instead reports held-out SWE-bench Pass@1 of 60.4%, versus 51.6% for its seed (Zhang et al., 2026e). These results test different forms of reuse and should not be compared as a common measure of generality. Cross-domain evidence also exists: ADAS transfers agents from mathematics to other reasoning tasks, and Multi-Task Self-Harness evaluates a frozen shared harness across benchmark mixtures (Hu et al., 2025; Xu, 2026). The remaining question is which modifications transfer unchanged, under which task and model shifts, rather than whether any cross-domain evaluation exists.

Across backbone models. Generalization across backbone models is also uncertain: because a harness mediates the behavior of a particular model, an improvement discovered with one model may exploit its specific reasoning patterns, failure modes, or tool-use behaviors and fail to benefit another. Some harnesses nonetheless transfer (Table 6): RRSI Xia et al. (2026) raises Terminal-Bench 2.1 accuracy from 11.2 to 14.6 on an unseen model. AHE Lin et al. (2026) gains 5.1–10.1 points across alternate model families. Transfer is nevertheless not guaranteed. In Evo-Harness Wei et al. (2026b), skills evolved by Claude Sonnet 4.5 improve a stronger Opus 4.7 solver on unseen tasks from 68.8 to 73.4, yet the same weaker Sonnet solver gains nothing from evolved skills, whether evolved by itself or by Opus (55.3 and 55.7 vs. 58.0 without evolution), suggesting that the benefit of evolved harness components depends on the capability of the model using them. This leaves open when a harness should transfer and when it should specialize to a particular backbone.

Future Directions. Future work should therefore optimize for mechanisms that transfer across task distributions, domains, and base models:

- **Reuse-controlled acceptance.** Reused acceptance sets risk adaptive overfitting. HarnessLens already uses fresh confirmation batches (Xu et al., 2026b); evaluate how refresh policies and reusable holdouts control error over many revisions.

Method	Held-out tasks	Cross-benchmark	Cross-domain	Cross-model
RRSI Xia et al. (2026)	Regularization	Regularization	–	Eval
HarnessCompass Zhang et al. (2026e)	Generalization gate	–	–	Eval
AHE Lin et al. (2026)	–	Eval	–	Eval
Meta-Harness Lee et al. (2026b)	Eval	Eval	–	Eval
Evo-Harness Wei et al. (2026b)	Skill abstraction	–	–	Eval

Table 6: Selected transfer configurations. "Eval" denotes reported evaluation; other entries identify associated design mechanisms, not isolated evidence of their causal effect. "–" means that unchanged transfer in that setting is not established by the configuration summarized here. This selection is not an exhaustive inventory of each paper or of cross-domain studies.

- **Systematic transfer evaluation.** Extend existing transfer studies with frozen harnesses, explicit task shifts, and matched resources across backbone families, scales, and domains.
- **Transfer-aware selection.** Compare selection on mixed tasks with independent transfer evaluation to distinguish selection gains from generalization.
- **Core–adapter decomposition.** A reusable harness core could be separated from lightweight task- and model-specific adapters, with principled rules for which components belong to each.

H.1.2 Reliable and Cost-Aware Measurement

Evaluation noise. Agent trajectories are stochastic, so single-attempt success does not establish repeatability. Thinkingbox reports GPT-5.4 at 65.36% pass@1 and 30.62% pass^{20} (Li et al., 2026c, Table 5; Appendix C.7, v4). The latter is a plug-in estimate of 20-fold success probability, not the fraction of tasks succeeding in all 20 observed trials. In a different setting, self-judged harness revision lowers GPT-5.4 from 75.3% to 69.7% pass@1 on Terminal-Bench 2.1 (Wang et al., 2026e, Table 1, v3). Acceptance mechanisms differ (Table 7): Self-Harness uses a two-split regression gate and reports whole-system improvements of 14.2–21.4 percentage points across three models on Terminal-Bench 2.0 (Zhang et al., 2026b). Its gate reads held-out scores, although held-out traces are hidden; these scores therefore participate in selection. SAGE uses a one-sided paired statistical test, while HarnessBank combines activation checks with paired significance tests (Wang et al., 2026d; Luo et al., 2026e). Such per-candidate safeguards should be distinguished from control of accumulated selection error across many adaptive comparisons.
Resource use. Performance must be interpreted alongside resource use. On two models and 50 Terminal-Bench Pro tasks, the Scaffold Effect study reports up to 40× variation in tokens per solved task; paired pass-rate differences are 0–8 percentage points and mostly have bootstrap intervals including zero (Vats and Golev, 2026). Under a matched $K = 5$ budget without unit tests, Wang et al. (2026e, Table 1, v3) report mean Terminal-Bench 2.1 pass@1 of 72.3% for parallel sampling, 67.4% for harness evolution, and 71.8% for harness scaling. These conditions matter: the result is not a universal ranking of optimization methods. MoMHa's synthetic-domain experiment compares joint optimization with an accuracy-then-tokens proposer (0.611 versus 0.584 mean accuracy, with 95 fewer tokens per example); its separate real-benchmark experiment compares against DSPy (0.539 versus 0.542, at 672 versus 2,229 tokens) (Mukherjee and Tanjim, 2026). In RRSI's workspace experiment, removing acceptance-side regularizers lowers the transfer average from 43.6 to 41.0 and raises target-policy tokens per trial from 2.42M to 3.59M (Xia et al., 2026). Existing objectives already include tokens, runtime, or safety (Robeyns et al., 2025; Mukherjee and Tanjim, 2026; Liu et al., 2026e). The reporting gap is consistent accounting for

Method	Evaluation noise	Resource use
Self-Harness Zhang et al. (2026b)	Two-split regression gate	–
MoMHa Mukherjee and Tanjim (2026)	–	Joint accuracy-safety-token objective
RRSI Xia et al. (2026)	Stability-aware acceptance	Cost-gated acceptance
SAGE Wang et al. (2026d)	Paired statistical acceptance test	Equal-budget comparison

Table 7: Selected mechanisms for reliable evaluation of harness improvement. "–" denotes no dedicated mechanism summarized in that cell, not absence of all analysis of that issue in the paper.

proposal, validation, and deployment costs, with ablations separating mechanism changes from additional computation.

Future Directions. Conceptually, an observed performance gain may reflect mechanism improvement, evaluation noise, or additional computation. Future systems therefore need more statistically principled promotion rules and evaluation:

- **Error-controlled promotion.** Extend per-candidate statistical gates to repeated adaptive selection, and measure error rates and power through controlled ablations.
- **Sequential evaluation.** Compare uncertainty-driven replication with existing adaptive allocation, such as HGM (Wang et al., 2026b), under a common total budget.
- **Cost-aware reporting and selection.** Make selection over both success and cost, including tokens, tool calls, and wall-clock time, standard practice, and compare against compute-matched baselines.

H.2 Attribute Gains before Expanding Search

H.2.1 Attribution of Harness Improvements

Within a generation. An increase in task success does not identify which interacting edit caused the gain. Meta-Harness's classification ablation raises median candidate accuracy from 34.6 to 50.0 when the proposer receives raw traces in addition to code and scores (Lee et al., 2026b, Table 3). This supports the value of proposer evidence, not the causal effect of an individual target-harness modification. In the adjacent setting of training-data research, RSIBench-Data links data decisions to checkpoint outcomes but explicitly labels its cross-case mechanisms as process observations rather than single-factor ablations (Meng et al., 2026). Target-harness methods do perform controlled checks: MemCodex isolates edits and uses paired comparisons, HarnessBank verifies activation and gains, and RRSI tests pruning (Wang and Liu, 2026; Luo et al., 2026e; Xia et al., 2026). These checks provide more than traceability, yet do not automatically separate all interactions within a multi-component revision.

Across generations. A currently neutral edit may enable a useful descendant, while an initially useful component can become redundant. DGM's ablation without open-ended archive exploration reaches 23.0% on SWE-bench, compared with 50.0% for the full system (Zhang et al., 2026c); this supports the value of preserving alternative ancestry, not the contribution of each archived edit. RRSI revisits retained modifications through pruning (Xia et al., 2026). The unresolved distinction is between an edit's current marginal value and its historical role in enabling later discoveries. Archive records and rollback checks support this analysis but do not themselves identify that long-term causal contribution.

Future Directions. Future work should move from failure-conditioned proposal toward counterfactual and interaction-aware attribution:

- **Counterfactual attribution.** Estimate each accepted edit's effect through selective ablation or rollback and paired evaluation of alternative descendants.
- **Interaction-aware attribution.** Use factorial or approximate Shapley-style analyses to estimate the contributions of edit combinations, while respecting dependencies among edits.

Method	Failure diagnosis	Edit traceability	Historical checks / reuse
Self-Harness Zhang et al. (2026b)	Verifier-grounded failure clustering	Minimal targeted edits	–
Meta-Harness Lee et al. (2026b)	Trace-conditioned proposal	Full candidate history	–
RRSI (Xia et al., 2026)	Evidence-aware attribution	Annealed edit budget	Pruning of stale edits
DGM (Zhang et al., 2026c)	Log-based self-diagnosis	One proposed feature per child	Archive as stepping stones

Table 8: Mechanisms supporting diagnosis, traceability, and historical analysis. Archive reuse and pruning do not by themselves establish causal attribution. "–" indicates no dedicated historical check in the configuration summarized.

Method	Editable space	Search guidance	Meta-improvement
Self-Harness Zhang et al. (2026b)	Harness components	Weakness-guided minimal edits	–
Meta-Harness Lee et al. (2026b)	Harness code	Traces + full candidate history	–
RRSI Xia et al. (2026)	Declared editable harness modules	Annealed edit budget + history-based exploration	–
STOP Zelikman et al. (2024)	Improver scaffold	Downstream gains of refined programs	Improver rewrites itself
DGM Zhang et al. (2026c)	Agent code	Open-ended archive search	Self-modifying agent

Table 9: Search strategies in self-improving agent and harness systems. "–" indicates that the mechanism producing modifications is not itself explicitly evolved.

- **Retrospective attribution.** Value edits both for resolving the failures that motivated them and for enabling or interfering with later improvements.

H.2.2 Defining the Search Space

Editable scope. Harnesses expose heterogeneous editable objects, from prompts to tools, memory procedures, and executable control flow (Table 9). Meta-Harness permits code-level search while making large execution records selectively readable (Lee et al., 2026b). Its Table 1 estimates approximately 10M tokens of diagnostic context *generated by one evaluation* in the largest setting; this is not the proposer's consumed context or its per-iteration token bill. Self-Harness and RRSI instead emphasize targeted edits and bounded revision (Zhang et al., 2026b; Xia et al., 2026). Structured localization already exists: AHE's reports associate failures with editable modules, and HARDE separates detection and intervention mechanisms (Lin et al., 2026; Liu et al., 2026e). The open question is how to adapt edit scope to evidence: when a local repair suffices, and when broader restructuring justifies its extra search and verification cost.
Challenges: Harness meta-improvement. Recent work goes beyond searching over harnesses by also improving the mechanism used to propose harness modifications (Table 9). STOP (Zelikman et al., 2024) recursively rewrites the scaffold performing improvement, while in DGM (Zhang et al., 2026c) the agent being improved also performs the modifications. DGM reports 39.0% on SWE-bench in its no-self-improvement ablation versus 50.0% for the full system (Zhang et al., 2026c); this is a configuration-level comparison, not an isolated effect of one rewrite. More generally, histories of successful and unsuccessful edits could be used to learn which modifications are promising for particular failure patterns. However, such histories carry noisy outcome labels and selection biases from previous generations, so a proposer learned from them may overfit to the run that produced it rather than transfer to new settings.
Future Directions. Future work on searching the harness space could pursue:

Method	Harness adaptation	Model update	Harness-to-model signal	Weight update tested?
RHI (Lee et al., 2026a)	Pairwise-refined loop description	–	Traces proposed for training	×
HELIX (Fan and Huang, 2026)	Harness portfolio	–	Labeled SFT, critic, filter, and preference data	×
SIA (Hebbar et al., 2026)	Tools, prompts, retries, search	RL updates	Rollouts under evolved scaffold	✓
Continual Harness Karten et al. (2026)	Prompts, skills, memory, sub-agents	Teacher-relabeled updates	Relabeled low-reward rollouts	✓

Table 10: Connections between harness adaptation and model learning. "–" indicates that model weights remain fixed. The final column records whether model training was tested, not whether recursive improvement was established. RHI proposes using improved trajectories for training; HELIX constructs labeled records. Neither tests a subsequent model update in the reported experiments.

- **Hierarchical search.** First identify the subsystem most likely responsible for a failure, such as planning, memory, tool use, or verification, then search for edits within it.
- **Hybrid proposal operators.** Use structured mutation operators as efficient priors for common changes, and fall back to open-ended code generation when they are insufficient.
- **Self-improving proposers.** Learn proposal mechanisms from edit histories while correcting for noisy outcomes and selection bias, so that improvements to the proposer transfer beyond the run that produced them.

H.2.3 Model–Harness Co-Evolution

Separating harness and model updates. Harness revision can change the trajectories available for model learning, while weight updates change which harness mechanisms remain useful (Table 10). Evidence for the two directions must be separated. RHI demonstrates improved execution with lower reasoning effort; it proposes resulting traces as training data but does not test a subsequent weight update (Lee et al., 2026a). HELIX's 65-candidate experiment reports 4.0% more coverage than Pi for its best fixed harness and up to 58.0% more for the complete *post-hoc portfolio*. A separate 200-slot sibling slice yields 438 verified training records, without demonstrating gains from training a model on them (Fan and Huang, 2026). SIA does compare harness-only and combined updates: adding weight learning raises LawBench accuracy from 50.0% to 70.1% and reduces TriMul runtime from 12,483 to 1,017 μs (Hebbar et al., 2026). Co-Harness and WHALE likewise couple harness search with model updates, while Continual Harness uses teacher-relabeled trajectories for online learning (Chen et al., 2026e; Kim et al., 2026a; Karten et al., 2026). These studies establish different forms of coupling; they should not be described as lacking all component comparisons. The remaining challenge is selective internalization: identify which discovered behavior benefits from weight learning, which should remain an external tool or control rule, and how to detect scaffolding made redundant by the updated model.

Future Directions. Future work should therefore study controlled model–harness co-evolution:

- **Adaptive update schedules.** Decide when to switch between harness and weight updates based on measured progress, while keeping training trajectories diverse enough to avoid reinforcing narrow strategies.
- **Cross-generation testing.** Evaluate old and new models with old and new harnesses to separate gains due to the model, the harness, and their co-adaptation.

Work	External constraints	Revalidation / recovery	Lineage safety
AHE (Lin et al., 2026)	Read-only verifier and config	–	–
DGM (Zhang et al., 2026c)	Sandbox, time limits, restricted web access, human oversight	–	Traceable archive (recording only)
Safety Must Survive Zhang et al. (2026h)	–	Revalidation of incumbents + validated rollback	Founder-based editing

Table 11: Safety mechanisms in recursive self-improvement. "–" indicates that the corresponding mechanism is not a primary focus of the work.

- **Internalization versus externalization.** Determine which discovered behaviors should be internalized into model weights and which should remain externalized in tools, memory, or control flow, and detect when internalized behaviors make scaffolding redundant.

H.3 Preserve Constraints across Generations

Safety along an improvement lineage. Harness self-evolution introduces a distinctive safety challenge: safety properties that hold for the initial agent may not persist after repeated self-modification. An accepted update can weaken behavioral constraints or introduce vulnerabilities, which then become part of the starting point for later generations. Safety must therefore be considered over the evolutionary trajectory rather than only for an individual harness snapshot. Following the evaluation levels and governance principles of Gong et al. (2026), we compare existing mechanisms along three dimensions: external constraints, revalidation and recovery, and lineage safety (Table 11). External constraints alone can be circumvented: in DGM (Zhang et al., 2026c), hiding the hallucination-detection functions from the agent reduced objective hacking but did not eliminate it, as one agent removed the logging of tool-use markers on which the check relied. In controlled authorization tasks, Safety Must Survive shows that current revalidation can select correct historical alternatives overlooked by stale scores. In its original-editor cohort of 42 failed roots, six generations of founder-based editing leave one unsafe endpoint, compared with 14 under current-code editing (Zhang et al., 2026h, Figure 4). The work jointly examines revalidation, fallback, and repair ancestry; safety here means absence of unauthorized effects under the specified checks. Extending these controls to broader agents requires tracking indirect dependencies, detecting inherited violations under changing checks, and preserving useful capabilities during recovery.
Future Directions. Future systems should therefore maintain persistent safety invariants and explicit recovery mechanisms throughout evolution:

- **Specifying the protected boundary.** Develop principled methods for deciding which components must remain immutable, including indirect dependencies, such as logging, through which protected checks can be bypassed.
- **Scalable revalidation.** Determine which accepted components to recheck, and when, so that revalidation remains affordable over long-horizon evolution.
- **Lineage-level recovery.** Detect and remove unsafe changes inherited across generations, and design edits whose effects remain reversible.
- **Integrated safeguards.** Combine protected boundaries, revalidation, and recovery within a single self-improving system.

The central open problem is how to preserve such non-bypassable guarantees without restricting the editable space so strongly that meaningful self-improvement becomes impossible.