BEYOND SELF-EVOLUTION: META-EVOLVING AGENTS FOR CROSS-TASK AGENT IMPROVEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Agents are increasingly improved by other agents. We study meta-evolution as a way to make this process more effective across task environments: an improver learns how to make effective changes from its accumulated experience of modifying agents. Our central question is whether this experience yields reusable improvement strategies that transfer to unseen tasks. We propose METAEVOFORGE, which couples task-level evolution of TASKAGENTs with the self-evolution of an independent improver, METAAGENT, while keeping all language-model weights fixed. Using asynchronous tree search across 62 training task environments, METAEVOFORGE collects 768 improvement pairs per training run. Each pair records a scoped package edit and the parent and child agents' behavior on matched tasks. The shared METAAGENT consolidates this evidence across environments to update its persistent improvement policy. We evaluate evolved and unevolved improvers on held-out task suites, starting from the same root agent under a fixed search budget. Evolved improvers yield task-dependent gains in final quality and discover high-scoring candidates with fewer tokens. These findings show that improvement experience accumulated at scale can itself transfer across task environments.

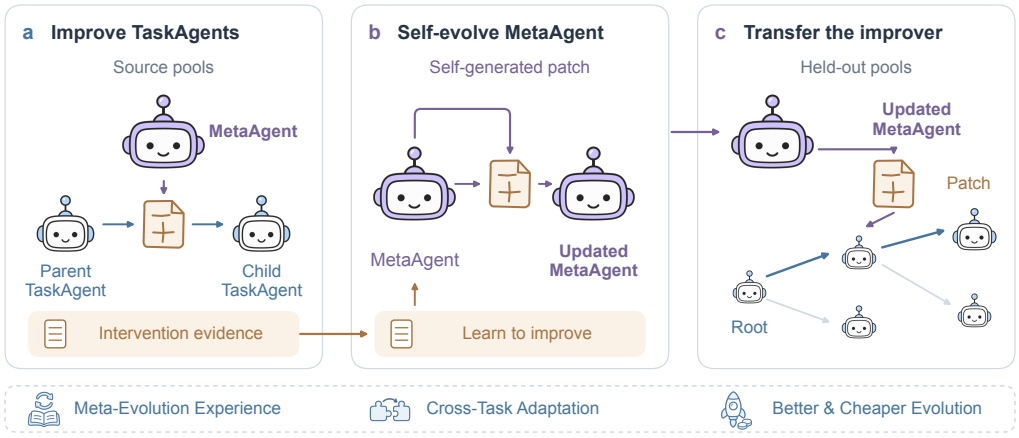


Figure 1: **Learning a transferable improver.** (a) A METAAGENT proposes TaskAgent package edits and collects paired parent–patch–child evidence from source pools. (b) It uses this experience to revise its own package. (c) The updated MetaAgent proposes edits for TaskAgent evolution in held-out pools. Bottom icons summarize meta-evolution experience, cross-task adaptation, and better and cheaper evolution. The tree is conceptual; experiments here evaluate transfer across task pools.

1 INTRODUCTION

Progress toward recursive self-improvement (RSI) requires AI systems to become increasingly effective at improving their successors (Jiang et al., 2026). Carrying this enhanced capability into subsequent cycles creates the potential for accelerating evolution: better improvers can produce stronger systems under the same budget. Recent advances in agent evolution (Park et al., 2026; Zhang et al.,

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

2025a) and learned editing policies (Shao et al., 2026) provide a foundation for studying this capability. We investigate whether it can develop through meta-evolution, in which an improver learns from earlier interventions to improve how it modifies agents. We ask whether this learning yields reusable improvement strategies for new task environments.

A better agent is not necessarily a better improver (Jiang et al., 2026; Zhang et al., 2026b). We distinguish the TASKAGENT, which performs tasks, from the METAAGENT, which proposes changes to it. The learning target is the METAAGENT's improvement policy: how it identifies a failure, chooses an edit, and uses the outcome to guide later proposals. Recent work has progressed from optimizing agent harnesses (Park et al., 2026) to learning editing policies and evolution-oriented meta-skills (Shao et al., 2026; Wu et al., 2026a). AEvo and MetaSkill-Evolve adapt the improvement process itself (Wang et al., 2026; Zhang et al., 2026c), while Hyperagents demonstrates transferable self-improvement through jointly evolving task and meta agents (Zhang et al., 2026b). Building on these advances, we ask how experience from separately evolving TASKAGENT across environments can be consolidated into a shared, reusable improver. These advances motivate our focus on an independently evolving METAAGENT that accumulates intervention experience across separate TASKAGENT lineages.

We instantiate this design in METAEVOFORGE, which couples TASKAGENT evolution with a shared METAAGENT's self-evolution (Figure 1). Either role can use any language-model-based agent, with model weights frozen during evolution. A unified task-pool protocol supplies execution evidence across environments. Each edit links parent behavior, a package modification, and child behavior on matched tasks. The METAAGENT uses these records to revise its own package, accumulating reusable improvement strategies.

To expose the improver to diverse tasks and assess transfer, we construct 69 task pools from public benchmarks and datasets. We train a shared METAAGENT on 62 pools and reserve seven pools for evaluation. On the held-out pools, the trained METAAGENT improves TASKAGENT performance faster and reaches a higher final score than its untrained initialization. A trained METAAGENT also evolves a Qwen3.6-35B-A3B-based TASKAGENT to performance comparable with that of an unevolved DeepSeek-V4-Flash-based TASKAGENT. We observe gains from METAAGENT training with two METAAGENT base models. Case studies and search trajectories show how its choice of edits changes during training, providing evidence of a learned policy for agent improvement.

Our contributions are:

- We introduce a meta-evolution framework that updates a persistent METAAGENT from parent–edit–child evidence collected across task pools.
- Across 62 training pools and seven held-out pools, we find that a trained METAAGENT generalizes its learned agent-evolution policy across held-out task pools, enables the METAAGENT to improve its TASKAGENT better with fewer tokens.
- Through case studies and trajectory analyses, we characterize how the METAAGENT's editing behavior changes during training, providing evidence that it acquires a meta-level capability for agent evolution.

2 RELATED WORK

Executable agents and harness evolution. Agent design can itself be searched. ADAS uses a fixed meta-agent to generate code-defined agents, whereas the Darwin Gödel Machine lets an agent edit and test its own implementation (Hu et al., 2025; Zhang et al., 2025a). Meta-Harness, AHE, and AutoSaddler extend this idea to model harnesses, using execution traces to guide code changes (Lee et al., 2026b; Lin et al., 2026; Park et al., 2026). Work in this line also varies in how candidate changes are proposed, selected, and evaluated. StarHarness separates proposal, selection, and held-out evaluation; ModularRSI evolves benchmark-disjoint modules before integration; RRSI regularizes proposal and selection; and SIFT uses pairwise patch judgments to reserve task evaluation for promising candidates (Esakkiraja et al., 2026; Fu et al., 2026; Wu et al., 2026b; Xia et al., 2026). These systems optimize the task-facing agent or harness. We study whether the improver can learn across task-agent lineages.

Learning to improve. Some work trains the editor itself. Harness-R1 learns runtime editing from failure trajectories; MetaEvolve trains iterative program refinement; and JIT-Agent learns task-conditioned harness generation and repair (Shao et al., 2026; Wu et al., 2026a; Zhang et al., 2026a). MetaEvo trains principle abstraction and reuses the resulting principles on later tasks (Ren et al., 2026). Other work changes the improvement mechanism. AEvo edits the procedure controlling future search, and Hierarchical Self-Improvement adds a meta-evolver that rewrites the evolver's strategy (Zhang et al., 2026c; Zhou, 2026). MetaSkill-Evolve maintains task and meta-skills on different timescales, while MCE co-evolves context-engineering skills and context artifacts (Wang et al., 2026; Ye et al., 2026). MetaRSI schedules operators over data, harnesses, and model updates (Tan et al., 2026); MEGA stores optimization knowledge in a Wisdom Graph (Lee et al., 2026a); and Meta^n recursively applies a fixed meta-operation (Kim et al., 2026).

Hyperagents is closest to our setting: both task and meta programs are editable, and meta-level improvements transfer across domains (Zhang et al., 2026b). Our METAAGENT is instead shared across separate TASKAGENT lineages and learns from paired parent–edit–child outcomes.

3 PROBLEM FORMULATION

Let $\mathcal{P}_{\text{train}}$ denote the task pools used to train the METAAGENT, and let $\mathcal{P}_{\text{test}}$ denote the held-out target pools used for evaluation. Each pool specifies a set of task instances, an execution environment, and a host-side evaluator. For a pool P, let p denote a sampled task subset used to evaluate a parent and its child on matched tasks.

For each TASKAGENT package A, we define an editable surface that specifies which components the METAAGENT may modify. The host keeps the evaluator, hidden answers, sandbox permissions, and task split fixed. Given execution evidence from the training tasks, an independent METAAGENT M proposes a patch δ that transforms the package at parent node u into the package at child node v:

$$A_v = A_u + \delta, \qquad \Delta J = J(A_v; p) - J(A_u; p), \tag{1}$$

where $J(A; p)$ is the task objective for package A on subset p. Whenever the pool permits paired evaluation, we evaluate the parent and child on the same subset.

The learning objective is to improve the METAAGENT M's proposal policy from its experience of modifying TASKAGENTs across $\mathcal{P}_{\text{train}}$. We represent its proposal role abstractly as

$$(A_u, p, R_u, S_u) \xrightarrow{M} \delta,$$

where R_u denotes the parent package A_u's execution record on p and S_u its aggregate validation score. Learning aims to improve this mapping so that the permitted patches δ it proposes yield larger performance gains ΔJ, as defined above. Language-model weights remain fixed throughout learning. We study whether this learned proposal policy transfers to unseen pools $P \in \mathcal{P}_{\text{test}}$. Starting from the same root A_0 under a common budget B of attempted child slots, we examine how training affects improvement progress and final TASKAGENT quality relative to the untrained improver M_0, and which learned editing behaviors support or limit this transfer.

Research questions. We ask whether a METAAGENT trained across task pools transfers to disjoint test pools and outperforms the root and the same untrained METAAGENT (**RQ1**); how the token cost of downstream adaptation relates to final TASKAGENT quality, and how this trade-off compares with external improvers (**RQ2**); and which learned improvement behaviors account for successes and regressions, including the limits of transfer (**RQ3**).

4 METHOD

METAEVOFORGE has two linked processes. Within each task pool, a shared METAAGENT proposes edits to a version tree of TASKAGENT packages. Parent and child versions are evaluated on matched task samples, and each edit is recorded with its behavioral outcome. Across pools, these intervention records are used to update the METAAGENT's own package. Model weights remain fixed throughout evolution. Figure 2 illustrates the two processes, and Algorithm 1 summarizes them.

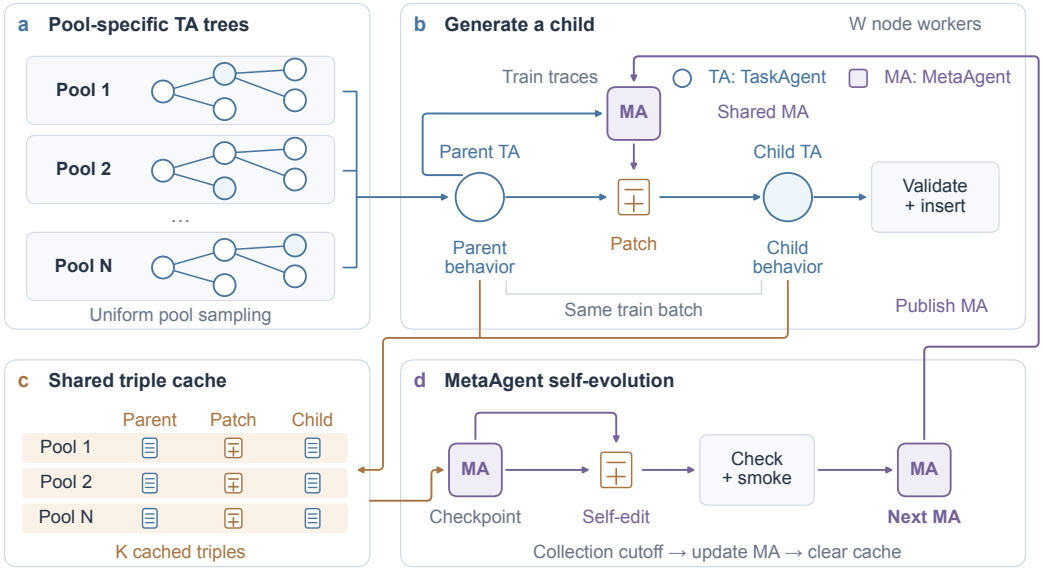


Figure 2: **Multi-pool evolution with a shared MetaAgent.** (a) Eligible pools are sampled uniformly; each maintains a separate TaskAgent (TA, circle) tree rooted at the common package. (b) Up to W node jobs use the shared MetaAgent (MA, rounded square) to propose valid edits, evaluate parent and child on the same training batch, and insert validation-scored children into their originating trees. (c) Paired behaviors and patches accumulate in a shared cache with pool provenance and aggregate validation scores; rows are illustrative, not quotas. (d) At K records, collection of outstanding jobs is cut off and the MA self-edits from a checkpoint. Package and proposal-smoke checks determine acceptance, retention, or rollback, not performance gains. The resulting MA is reused after clearing the cache. The cutoff discards late results without requiring running executions to stop immediately.

4.1 WITHIN-POOL TASKAGENT EVOLUTION

Each pool maintains a version tree rooted at A_0. A parent u is sampled with validation-score-based weights, and a training subset p is sampled from the pool. Evaluating A_u on p yields R_u, a training execution trace that includes reference answers. The METAAGENT uses these artifacts to propose a patch δ. A valid patch produces A_v, which is evaluated on the same p to obtain R_v. Parent and child are also scored on the pool's shared validation set, yielding S_u and S_v. We record the intervention as

$$\Delta_{p,u,v} = (p, u, \delta, v, R_u, R_v, \Delta J, S_u, S_v). \tag{2}$$

Because the METAAGENT can inspect reference answers in R_u, validation inputs and answers are withheld from it; only aggregate validation scores are available. This tests whether an edit's effect extends beyond p and limits overfitting to the sampled training tasks. The tree retains each valid child, including regressions, with its package snapshot and parent link. Edits may change agent-owned prompts, skills, memory, or executable components; the evaluator, task splits, harness implementation, and execution permissions remain fixed.

Algorithm 1: Learning a transferable METAAGENT (overview)

Input: training pools $\mathcal{P}_{\text{train}}$, root A_0, initial improver M_0
initialize version trees $\{T_P\}$, cache $\mathcal{C} \leftarrow \emptyset$, and $M \leftarrow M_0$
while update and node budgets remain **do**
  choose eligible pool P, parent $u \in T_P$, and training subset p
  $R_u \leftarrow \text{EVALUATE}(A_u, p)$; skip mutation if the parent is saturated
  $(\delta, A_v) \leftarrow \text{PROPOSEVALIDEDIT}(M, A_u, p, R_u, S_u)$
  if no valid changed package is obtained **then continue**
  $R_v \leftarrow \text{EVALUATE}(A_v, p)$; $S_v \leftarrow \text{VALIDATIONSCORE}(A_v, P)$
  form $\Delta_{p,u,v}$; add v to T_P and the intervention to $\mathcal{C}$
  if $|\mathcal{C}| \geq K$ **then** $M \leftarrow \text{SELFUPDATE}(M, \mathcal{C})$; $\mathcal{C} \leftarrow \emptyset$
end while; **return** accepted METAAGENT M and version trees

4.2 CROSS-POOL METAAGENT LEARNING

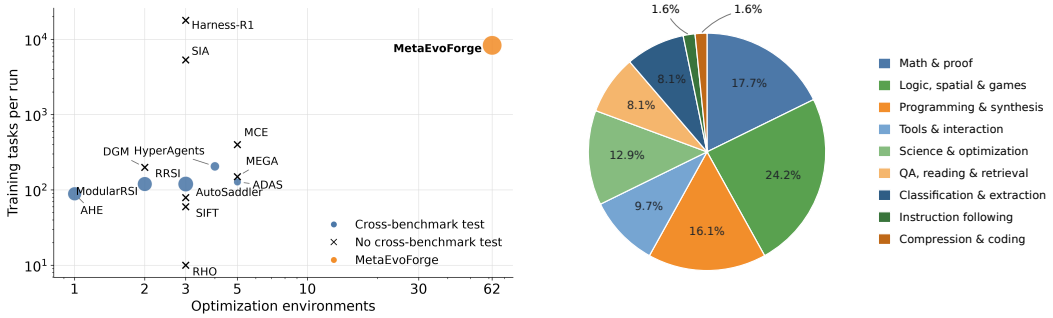
Intervention records from the training pools accumulate in a shared cache $\mathcal{C}$ (Figure 2c). Their task subsets and package lineages retain the context of each outcome. After K records, the METAAGENT examines the patches, training traces, and aggregate validation scores and proposes changes to its own policy files, skills, memory, or executable components (Algorithm 1, lines 3–9). The cache is then cleared, while the pool-specific TASKAGENT trees persist.

During each collection phase, eligible pool trees generate nodes concurrently under the current METAAGENT version. Each completed intervention links an edit to the parent's observed behavior and the child's measured outcome, giving self-update evidence from different task contexts. Once the cache reaches K, the scheduler stops dispatching nodes and excludes unfinished jobs from the batch. The METAAGENT then revises its proposal procedure before node generation resumes. Repeating this cycle lets outcomes across pools inform how the METAAGENT diagnoses and edits future TASKAGENTS. Detailed queue and node pseudocode, retry rules, edit boundaries, and self-update checks appear in Appendix B.

5 EXPERIMENTAL SETUP

Task pools and data partitions. We train the METAAGENT on 62 source pools and evaluate it on seven different target pools. A TaskPackage can contain a question, an episode, or a batch of examples; throughout, task counts refer to package counts rather than the numbers of upstream benchmark examples or model calls. Fixed manifests separate training or adaptation, validation, and final-test packages. Target-pool adaptation packages never enter METAAGENT training.

Beyond the number of training tasks, our setup differs from prior meta-evolving systems in both optimization breadth and task diversity. Figure 3 compares these dimensions and summarizes the composition of our source pools.



(a) Optimization breadth and training scale.

(b) Composition of the 62 source pools.

Figure 3: Scale and composition of the training data. Figure (a) compares the number of optimization environments and the maximum number of unique training tasks used in one run. Both axes use logarithmic scales. Circle area is proportional to the reported cross-benchmark test-set size, while black crosses denote methods without such evaluation.

The target pools are LawBench, Symptom2Disease, USPTO, Polyglot, DBBench, OptMATH, and FormalMATH-Lite. Their fixed validation partitions contain 12–100 packages per pool; their final-test partitions contain 99–300. The seven target pools contribute equally to the reported macro mean, regardless of their different test sizes.

Models, training, and checkpoints. The main runs use `deepseek-v4-flash-0731` (DeepSeek-AI, 2026a) and `glm-5.3-flash` (Z.ai, 2026), respectively, as their METAAGENT base models; both use `Qwen3.6-35B-A3B` as the TASKAGENT base model and share the source pools, DeepSeek Harness (DeepSeek-AI, 2026b), and CubeSandbox (Tencent Cloud, 2026). We choose DeepSeek Harness for its native support for hot-swapping harness components during on-

line evolution. We use CubeSandbox as a lightweight sandbox that isolates the execution of agentic tasks.

Both training runs use 32 METAAGENT self-update cycles. Each update is triggered by $K = 24$ collected parent–child records. Parent and child TASKAGENT are run once on a matched batch of eight tasks randomly sampled from the same pool. Source-pool trees allow at most 64 attempted child slots per pool. We retain the initial METAAGENT M_0 and checkpoints after cycles 8 (M_8), 16 (M_{16}), and 32 (M_{32}). A cycle counts an update attempt, including attempts that fail to produce a valid self-update patch and are rolled back.

Held-out search and final selection. For each METAAGENT checkpoint, target-pool search starts from the same initial TASKAGENT package. The budget is $B = 32$ attempted child slots per pool; a skipped attempt consumes a slot. Parent selection uses validation-score-weighted sampling. Parent and child are each run once on the same sampled batch of 16 adaptation packages. Each generated child is then evaluated once on that pool's full fixed validation partition. The highest-validation candidate is selected to be test on held-out pools; ties are broken by creation order.

Baseline improvers. We compare the trained METAAGENT with six external improvers: Hyperagents (Zhang et al., 2026b), Meta-Harness (Lee et al., 2026b), AHE (Lin et al., 2026), DGM (Zhang et al., 2025a), the harness-only SIA-H variant (Hebbar et al., 2026), and Retrospective Harness Optimization (RHO) (Pan et al., 2026). We do not train these baselines on our source pools. On each held-out pool, each compatible method starts from the same root and has access to the same number of TaskPackages. The resulting agents are evaluated on the same final-test tasks. For a controlled comparison, we adapt every baseline to DeepSeek Harness and CubeSandbox, using `deepseek-v4-flash-0731` for the improver role and `Qwen3.6-35B-A3B` for the TASKAGENT role.

6 RESULTS

6.1 DOES TRAINING IMPROVE HELD-OUT TASKAGENT EVOLUTION?

We ask whether a trained METAAGENT can evolve a stronger TASKAGENT on unseen pools than the same METAAGENT before training. Table 1 reports one held-out search per checkpoint and pool. Our checkpoint scores are final-test scores of validation-selected TASKAGENT.

Table 1: Final-test scores (%) of evolved Qwen TASKAGENTs on seven held-out pools. Our candidates are selected by validation; external methods use their own selection rules. Macro is the unweighted mean of all seven pool scores shown. † Meta-Harness is reported on six of the seven pools because its evolution algorithm is incompatible with the FormalMATH evaluator.

Improver	Law	S2D	USPTO	Poly	DB	Opt	Formal	Macro
Root TASKAGENT	65.00	66.04	8.00	14.46	62.33	5.00	12.00	33.26
DeepSeek METAAGENT								
M_0	63.00	81.60	18.00	24.10	64.67	17.00	37.00	43.62
M_8	63.00	74.06	20.00	29.52	71.67	30.00	46.00	47.75
M_{16}	66.00	77.83	**25.00**	30.72	**77.00**	**37.00**	31.00	**49.22**
M_{32}	67.00	73.58	22.00	30.72	76.00	26.00	44.00	48.47
GLM-5.3-Flash METAAGENT								
M_0	55.00	74.53	14.00	23.49	52.00	19.00	29.00	38.15
M_8	64.00	73.11	18.00	29.52	59.00	23.00	44.00	44.38
M_{16}	67.00	73.58	17.00	30.72	68.33	29.00	43.00	46.95
M_{32}	64.00	77.36	18.00	**31.93**	74.33	29.00	44.00	48.37
External improvers								
Hyperagents	65.00	**86.32**	20.00	28.31	43.67	24.00	14.00	40.19
Meta-Harness	47.00	76.42	13.00	12.65	30.67	23.00	–	33.79†
AHE	71.00	69.81	23.00	24.70	53.00	11.00	37.00	41.36
DGM	64.00	67.45	7.00	24.10	63.33	9.00	28.00	37.55
SIA-H	70.00	79.72	14.00	16.27	68.33	12.00	34.00	42.05
RHO	**72.00**	66.98	8.00	31.33	73.67	25.00	**48.00**	46.43

Across both base-model families, every trained checkpoint exceeds its own M_0 on the seven-pool macro (Table 1). The GLM trajectory gives the clearest contrast: M_{32} gains 10.22 points and im-

proves on every pool. DeepSeek is less monotonic; M_{16} slightly exceeds M_{32} overall, suggesting that meta-agent self-evolution had begun to saturate by M_{16}. The best trained checkpoints from both families exceed every external baseline with a complete seven-pool score on the macro mean, although individual baselines lead on particular pools. Taken together, the within-model gains suggest that the METAAGENT learns improvement strategies across diverse source pools that transfer to TASKAGENT evolution on held-out pools.

However, two exception cases show that evolution does not guarantee improvement. On S2D, M_0 developed a benchmark-specific guide and achieved 81.60. Later evolution accumulated additional exception rules, but these changes reduced M_{32}'s score to 73.58. Nevertheless, M_{32} outperformed M_0 on six of the seven held-out pools. External improvers also exhibit ineffective evolution cases. In one DBBench run, AHE introduced an automatic column checker after observing that the agent sometimes retrieved values from the wrong database column. The resulting candidate achieved higher training accuracy. On a validation task, however, the checker issued an incorrect warning, causing the agent to second-guess itself and replace a correct answer with an incorrect one (Appendix C.5).

6.2 COST OF FINDING A STRONG CANDIDATE

We compare adaptation cost with held-out performance in Figure 4. The horizontal axis aggregates accounted language-model tokens across the available held-out pools.

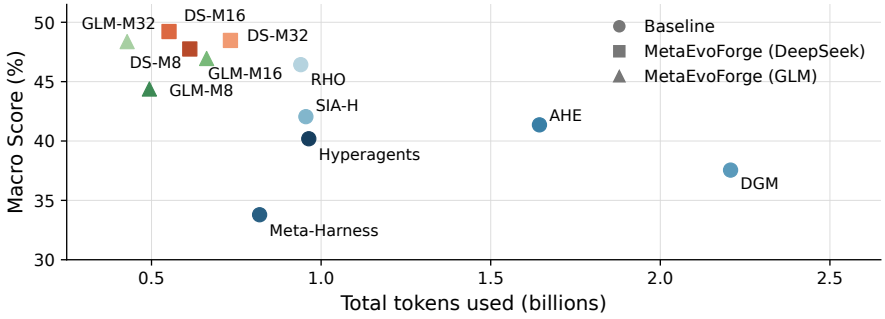


Figure 4: Total accounted language-model tokens versus held-out macro score.

MetaEvoForge checkpoints occupy the upper-left region of the plot. DS-M16 achieves the highest macro score of 49.22% with 0.55B accounted tokens, outperforming RHO, the strongest external improver at 46.43%, while using 41% fewer tokens than RHO's 0.94B. DeepSeek M_8 and M_{32} also exceed every external improver in macro score while using fewer accounted tokens. The GLM checkpoints use similarly low token totals and reach comparable macro scores. Together, MetaEvoForge can produce stronger TaskAgents without proportionally increasing adaptation cost.

6.3 PORTABILITY OF AN EVOLVED HARNESS ACROSS TASKAGENT MODELS

We compare four conditions on the seven held-out pools in Table 2: the Qwen TASKAGENT evolved by DeepSeek M_{32} (Q), an unevolved DeepSeek TASKAGENT (R), a direct swap of a Qwen-selected harness to the DeepSeek base model (S), and a DeepSeek TASKAGENT produced by a new M_{32} search (E).

A new M_{32} search with DeepSeek scores 5.52 points below the direct swap and 4.03 points below the DeepSeek root. This suggests that the evolution strategy learned with Qwen-based TASKAGENTs is sensitive to the TASKAGENT base model: in this run, it did not transfer effectively to a fresh DeepSeek search. Consistent with the non-monotonic checkpoint trend in Section 6.1, M_{32} may have become specialized to Qwen-based evolution, although one search cannot establish this mechanism. The direct swap gives a different result. Replacing only the evolved harness's base model yields a 1.49-point gain over the DeepSeek root and improves four of seven pools. The direct swap also exceeds the Qwen condition on the macro mean. These results suggest that part of the selected harness remains useful across base models, even when a fresh search with the same METAAGENT

Table 2: Final-test scores (%) on seven held-out pools. Q is Qwen evolved, R is DeepSeek root, S is the direct model swap, and E is new evolution with DeepSeek. Differences are in percentage points. Macro weights pools equally; comparisons across separate runs are descriptive.

Pool	Q	R	S	E	$Q-R$	$S-R$	$E-S$
LawBench	67.00	65.00	63.00	52.00	+2.00	−2.00	−11.00
S2D	73.58	75.94	64.15	64.15	−2.36	−11.79	0.00
USPTO	22.00	30.00	24.00	28.00	−8.00	−6.00	+4.00
Polyglot	30.72	40.36	47.59	40.96	−9.64	+7.23	−6.63
DBBench	76.00	60.67	78.67	71.67	+15.33	+18.00	−7.00
OptMATH	26.00	42.00	46.00	34.00	−16.00	+4.00	−12.00
FormalMATH	44.00	47.00	48.00	42.00	−3.00	+1.00	−6.00
Macro mean	48.47	51.57	53.06	47.54	−3.10	+1.49	−5.52

does not improve performance. The evolved Qwen TASKAGENT finishes 3.10 macro points below the unevolved DeepSeek TASKAGENT. In this setting, an agent evolved from the smaller Qwen model approaches the performance of an unevolved agent built on the larger DeepSeek model.

Together, these results suggest that our meta-learning approach can discover harnesses that retain useful behavior across base models, while its evolution strategy remains partly adapted to the model used during training. This combination allows an evolved smaller-model agent to approach the performance of an unevolved frontier-model agent, though a fresh search on a different base model does not necessarily yield the same benefit.

7 ANALYSIS

7.1 EVOLUTION OF THE METAAGENT PROPOSAL POLICY

We semantically annotate all 768 TASKAGENT patch proposals from 32 completed METAAGENT self-update cycles into three groups: contract/runtime edits, transferable task-family reasoning, and reference-specific labels, tokens, or behaviors. Labels are assigned from proposal intent and patch content without exposing update cycle or validation outcome. Figure 5 shows the resulting category mix and validation outcome together with the package surfaces modified over training.

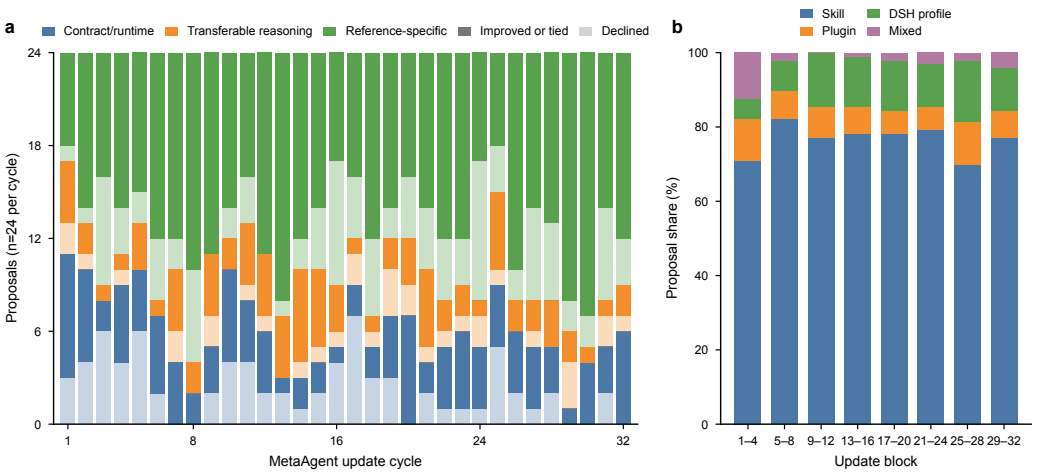


Figure 5: Proposal semantics, validation outcomes, and edit surfaces across 32 METAAGENT self-update cycles ($n = 768$). (a) Each cycle contains 24 completed proposals. Color denotes proposal semantics; saturated segments denote improved or tied validation scores and light segments denote decreased scores. (b) Mutually exclusive edit-surface categories aggregated over consecutive four-cycle blocks.

Contract/runtime edits decrease 31.8% in cycles 1–8 to 21.4% in cycles 25–32, while reference-specific edits increase from 55.7% to 65.1%. Transferable reasoning is not monotonic: its share peaks at 20.3% in cycles 9–16 and ends at 13.5%. Thus, the run shifts away from generic contract repair toward pool- or reference-specific adaptation, but does not show steady growth in broadly transferable task-family reasoning. Since each TASKAGENT is evolved within one pool, these reference-specific proposals provide effective local adaptation within the corresponding task environment.

7.2 TRAINING DYNAMICS ACROSS SOURCE POOLS

We next compare two complete training runs that use DeepSeek and GLM as the METAAGENT backend while sharing the same 62 source pools. Each run contains 32 self-update cycles and collects 24 interventions per cycle. At every cycle, we retain the best validation score observed so far in each pool. The resulting macro score rises from 45.69% to 74.91% for DeepSeek and 76.56% for GLM (Figure 6).

The pool-level view shows that gains remain heterogeneous across tasks and accumulate at different rates under the two backends (Figure 7). Overall, 56 of 62 pools improve under DeepSeek and 58 improve under GLM. Together with the proposal-category analysis, this pattern motivates examining which editing principles recur in successful interventions and which failure modes remain unresolved. Training trajectories and per-pool validation gains are shown in Appendix D.

These training trajectories summarize validation feedback used during optimization and complement the held-out checkpoint comparison in Section 6.1. Since TASKAGENT trees accumulate throughout training, the trajectory alone does not isolate changes in METAAGENT ability from additional search. We use it to contextualize the observed proposal policy and its concentration by edit surface.

7.3 THE METAAGENT LEARNS SKILL-CENTRIC PROPOSALS AND EVIDENCE-DRIVEN SELF-UPDATES

The system prompt is fixed outside the editable package in these runs, so the proposal distribution reflects choices among package-owned skills, plugins, and the DSH profile. Of 768 proposals, 588 (76.6%) modify skills only, 63 (8.2%) modify plugins only, 91 (11.8%) modify only the DSH profile, and 26 (3.4%) span multiple surfaces (Figure 5b). Counting mixed proposals on every surface they touch, 78.8% include a skill, 10.7% include executable plugin code, and 14.1% include the DSH profile.

Skill-only edits dominate every four-cycle block, but executable and profile edits persist throughout training. The common skill changes encode submission contracts, task-family procedures, and reusable reasoning routines, whereas plugin and profile changes alter runtime behavior or DSH composition. Thus the proposal policy strongly favors lower-risk declarative edits without treating the executable package surface as unavailable.

All accepted self-updates revise persistent proposal-policy skills rather than executable plugins or the DSH profile. Rather than storing task-specific answers, these updates refine how the METAAGENT interprets evidence and chooses subsequent edits. In particular, it learns to revisit false or overly broad guidance when gains observed in matched evaluations do not transfer, and to screen out zero-event or stalled runs before attributing an outcome to a patch. Self-update therefore changes the policy used to propose and assess later edits across pools.

8 CONCLUSION

We investigate a focused step toward AI systems that improve AI: whether a persistent METAAGENT can learn from earlier TASKAGENT interventions and then improve a new TASKAGENT better in held-out task environments. METAEVOFORGE makes that question testable through large scale source task pools and scoped parent–patch–child behavioral evidence. This work advances a practical path toward recursive self-improvement: consolidating experience across evolving agents into reusable strategies for improving future agents.

AI USE STATEMENT

During this research, generative AI tools were used to assist with language polishing and clarity of presentation, the retrieval and discovery of relevant literature, and code debugging. We did not use generative AI tools to draft sections of the manuscript, generate synthetic datasets, or prove mathematical claims.

The authors reviewed all AI-assisted work. AI-assisted text was checked, revised, and approved by the authors, and translations were checked against the authors' intended meaning. Reported numbers were checked against experimental records, citations against their original sources, and figures against the underlying data. AI-assisted code was read, executed, tested, and debugged by the authors to verify its correctness. The authors take full responsibility for the final content of this work, including all text, claims, code, and other artifacts produced with the aid of generative AI.

REFERENCES

Aider contributors. Aider Polyglot Benchmark, n.d. URL https://github.com/Aider-AI/polyglot-benchmark. Data/software resource; accessed 6 September 2026.

Chenxin An, Zhihui Xie, Xiaonan Li, Lei Li, Jun Zhang, Shansan Gong, Ming Zhong, Jingjing Xu, Xipeng Qiu, Mingxuan Wang, and Lingpeng Kong. POLARIS: A Post-Training Recipe for Scaling Reinforcement Learning on Advanced Reasoning Models, 2025. URL https://hkunlp.github.io/blog/2025/Polaris/.

Center for AI Safety, Scale AI, and HLE Contributors Consortium. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649:1139–1146, 2026. doi: 10.1038/s41586-025-09962-4. URL https://doi.org/10.1038/s41586-025-09962-4.

Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. ARC-AGI-2: A New Challenge for Frontier AI Reasoning Systems, 2026. URL https://arxiv.org/abs/2505.11831.

François Chollet. On the Measure of Intelligence, 2019. URL https://arxiv.org/abs/1911.01547.

Compfiles contributors. Compfiles: Catalog of Math Problems Formalized in Lean, n.d. URL https://github.com/dwrensha/compfiles. Data/software resource; accessed 6 September 2026.

DeepSeek-AI. DeepSeek-V4: Towards Highly Efficient Million-Token Context Intelligence, 2026a. URL https://arxiv.org/abs/2606.19348.

DeepSeek-AI. DeepSeek Harness: Everything is a Plugin. Software repository, 2026b. URL https://github.com/deepseek-ai/deepseek-harness. Accessed 6 September 2026.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378. Association for Computational Linguistics, 2019. doi: 10.18653/v1/N19-1246. URL https://aclanthology.org/N19-1246/.

Alexander Dunn, Qi Wang, Alex Ganose, Daniel Dopp, and Anubhav Jain. Benchmarking materials property prediction methods: the Matbench test set and Automatminer reference algorithm. *npj Computational Materials*, 6(1), 2020. doi: 10.1038/s41524-020-00406-3. URL http://dx.doi.org/10.1038/s41524-020-00406-3.

Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Guney, Volkan Cirik, and Kyunghyun Cho. SearchQA: A new Q&A dataset augmented with context from a search engine, 2017. URL https://arxiv.org/abs/1704.05179.

Esakkivel Esakkiraja, Denis Akhiyarov, Vikas Yadav, Sai Rajeswar, Patrice Bechard, Sridhar Nemala, and Sagar Davasam. StarHarness: Evolving Harnesses with Stratified Search for Enterprise Environments, 2026. URL https://arxiv.org/abs/2608.24804.

Zhiwei Fei, Xiaoyu Shen, Dawei Zhu, Fengzhe Zhou, Zhuo Han, Songyang Zhang, Kai Chen, Zongwen Shen, and Jidong Ge. LawBench: Benchmarking Legal Knowledge of Large Language Models, 2023. URL https://arxiv.org/abs/2309.16289.

Xinghong Fu, Aravinth Kulanthaivelu, and Yutaro Yamada. Self Improvement via Fast Tree-search, 2026. URL https://arxiv.org/abs/2609.19526.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. Omni-MATH: A Universal Olympiad Level Mathematic Benchmark For Large Language Models, 2024. URL https://arxiv.org/abs/2410.07985.

Shaona Ghosh, Prasoon Varshney, Makesh Narsimhan Sreedhar, Aishwarya Padmakumar, Traian Rebedea, Jibin Rajan Varghese, and Christopher Parisien. AEGIS2.0: A Diverse AI Safety Dataset and Risks Taxonomy for Alignment of LLM Guardrails. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5992–6026. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.306. URL https://aclanthology.org/2025.naacl-long.306/.

Aryan Gulati, Brando Miranda, Eric Chen, Emily Xia, Kai Fronsdal, Bruno De Moraes Dumont, and Sanmi Koyejo. Putnam-AXIOM: A Functional & Static Benchmark for Measuring Higher Level Mathematical Reasoning in LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 20723–20747. PMLR, 2025. URL https://proceedings.mlr.press/v267/gulati25a.html.

Shijie Han, Haoqiang Kang, Bo Jin, Xiao-Yang Liu, and Steve Y. Yang. XBRL Agent: Leveraging large language models for financial report analysis. In *Proceedings of the 5th ACM International Conference on AI in Finance*, pp. 856–864. Association for Computing Machinery, 2024. doi: 10.1145/3677052.3698614. URL https://doi.org/10.1145/3677052.3698614.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A Challenging Benchmark for Promoting AGI with Olympiad-Level Bilingual Multimodal Scientific Problems, 2024. URL https://arxiv.org/abs/2402.14008.

Prannay Hebbar, Yogendra Manawat, Samuel Verboomen, Alesia Ivanova, Selvam Palanimalai, Kunal Bhatia, and Vignesh Baskaran. SIA: Self improving AI with harness & weight updates, 2026. URL https://arxiv.org/abs/2605.27276.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021a. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c24cd76e1ce41366a4bbe8a49b02a028-Abstract-round2.html.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset, 2021b. URL https://arxiv.org/abs/2103.03874.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2408.08435.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. ORLM: A Customizable Framework in Training Large Models for Automated Optimization Modeling, 2025. URL https://arxiv.org/abs/2405.17743.

Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and Kang Liu. DA-Code: Agent Data Science Code Generation Benchmark for Large Language Models, 2024. URL https://arxiv.org/abs/2410.07331.

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, Xin Jiang, Shulei Xie, Yuanchun Zheng, Youbang Sun, Biqing Qi, Ning Ding, Kaiyan Zhang, and Bowen Zhou. Self-improving agents in the era of experience: A survey of self- to meta-evolution. *OpenReview Archive*, 2026. URL https://openreview.net/pdf?id=IUltZSgLMm.

Yichen Jiang, Shikha Bordia, Zheng Zhong, Charles Dognin, Maneesh Singh, and Mohit Bansal. HoVer: A Dataset for Many-Hop Fact Extraction And Claim Verification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 3441–3460. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.findings-emnlp.309. URL https://aclanthology.org/2020.findings-emnlp.309/.

Steven M. Kearnes, Michael R. Maser, Michael Wleklinski, Anton Kast, Abigail G. Doyle, Spencer D. Dreher, Joel M. Hawkins, Klavs F. Jensen, and Connor W. Coley. The Open Reaction Database. *Journal of the American Chemical Society*, 143(45):18820–18826, 2021. doi: 10.1021/jacs.1c09820. URL http://dx.doi.org/10.1021/jacs.1c09820.

Zae Myung Kim, Young-Jun Lee, Seungyeon Jwa, and Dongyeop Kang. Meta^n: Recursive Self-Improvement through Emergent Depth, 2026. URL https://arxiv.org/abs/2608.24735.

Jung Hwan Lee, Kyu Ho Lee, and Gwang Hoon Yoo. MEGA: Self-Evolving Agent Optimization Infrastructure via Wisdom Graph, 2026a. URL https://arxiv.org/abs/2608.10504.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses, 2026b. URL https://arxiv.org/abs/2603.28052.

Jinyang Li, Xiaolong Li, Ge Qu, Per Jacobsson, Bowen Qin, Binyuan Hui, Shuzheng Si, Nan Huo, Xiaohan Xu, Yue Zhang, Ziwei Tang, Yuanshuai Li, Florensia Widjaja, Xintong Zhu, Feige Zhou, Yongfeng Huang, Yannis Papakonstantinou, Fatma Ozcan, Chenhao Ma, and Reynold Cheng. SWE-SQL: Illuminating LLM Pathways to Solve User SQL Issues in Real-World Applications, 2025a. URL https://arxiv.org/abs/2506.18951.

Junlong Li, Daya Guo, Dejian Yang, Runxin Xu, Yu Wu, and Junxian He. CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction, 2025b. URL https://arxiv.org/abs/2502.07316.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs, 2023a. URL https://arxiv.org/abs/2304.08244.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. TACO: Topics in algorithmic COde generation dataset, 2023b. URL https://arxiv.org/abs/2312.14852.

Lichess. Lichess Open Puzzle Database, n.d. URL https://database.lichess.org/#puzzles. Data/software resource; accessed 6 September 2026.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. ZebraLogic: On the Scaling Limits of LLMs for Logical Reasoning, 2025. URL https://arxiv.org/abs/2502.01100.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses, 2026. URL https://arxiv.org/abs/2604.25850.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as Agents, 2025. URL https://arxiv.org/abs/2308.03688.

Lefteris Loukas, Manos Fergadiotis, Ilias Chalkidis, Eirini Spyropoulou, Prodromos Malakasiotis, Ion Androutsopoulos, and Georgios Paliouras. FiNER: Financial Numeric Entity Recognition for XBRL Tagging. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4419–4431. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.acl-long.303. URL https://aclanthology.org/2022.acl-long.303/.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. OptMATH: A Scalable Bidirectional Data Synthesis Framework for Optimization Modeling, 2025. URL https://arxiv.org/abs/2502.11102.

Matt Mahoney. Large Text Compression Benchmark, 2026. URL https://www.mattmahoney.net/dc/text.html. Updated 2 September 2026; accessed 6 September 2026.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

Wenbo Pan, Shujie Liu, Chin-Yew Lin, Jingying Zeng, Xianfeng Tang, Xiangyang Zhou, Yan Lu, and Xiaohua Jia. Evolving agents in the dark: Retrospective harness optimization via self-preference, 2026. URL https://arxiv.org/abs/2606.05922.

Sungho Park, Wonjoong Kim, Rongyuan Tan, Jue Zhang, Wook-Shin Han, Pengfei Gao, Chanyoung Park, Yongqiang Yao, Rao Fu, Elsie Nallipogu, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. AutoSaddler: Automatic harness optimization with durable updates from agent execution traces, 2026. URL https://arxiv.org/abs/2608.23041.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://arxiv.org/abs/2507.02833.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. NL4Opt Competition: Formulating Optimization Problems Based on Their Natural Language Descriptions. In *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pp. 189–203. PMLR, 2022. URL https://proceedings.mlr.press/v220/ramamonjison23a.html.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A Graduate-Level Google-Proof Q&A Benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Bowen Ren, Heyan Huang, Yinghao Li, and Yang Gao. MetaEvo: A Meta-Optimization Framework for Experience-Driven Agent Evolution, 2026. URL https://arxiv.org/abs/2606.07603.

Raimondas Sasnauskas, Yang Chen, Peter Collingbourne, Jeroen Ketema, Gratian Lup, Jubi Taneja, and John Regehr. Souper: A Synthesizing Superoptimizer, 2018. URL https://arxiv.org/abs/1711.04422.

Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic Superoptimization, 2012. URL https://arxiv.org/abs/1211.0557.

Max-Philipp B. Schrader. gym-sokoban, 2018. URL https://github.com/mpSchrader/gym-sokoban. Data/software resource; accessed 6 September 2026.

Shuai Shao, Kangning Zhang, Qingyao Li, Shijian Wang, Hao Wang, Wenxiang Jiao, Yuan Lu, Yi Guo, Weiwen Liu, and Weinan Zhang. Harness-R1: Learning to edit executable runtime harnesses from agent failure trajectories, 2026. URL https://arxiv.org/abs/2608.02276.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. Language models are multilingual chain-of-thought reasoners. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=fR3wGCk-IXp.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning, 2021. URL https://arxiv.org/abs/2010.03768.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. REASONING GYM: Reasoning Environments for Reinforcement Learning with Verifiable Rewards, 2025. URL https://arxiv.org/abs/2505.24760.

SyGuS Organization. Benchmarks for the Syntax-Guided Synthesis Competition, n.d. URL https://github.com/SyGuS-Org/benchmarks. Data/software resource; accessed 6 September 2026.

Zihan Tan, Leixin Sun, Zitong Shi, Yitao Liu, Jiajun Wu, Nathaniel Brooks, Jiaru Qian, Xiaoran Shang, Suyuan Huang, Yi Ding, Yangxu Liao, Mukai Li, Qiushi Sun, Shudong Liu, Xuankun Rong, Xiaohang Yu, Zhuo Chen, Hejia Geng, Chenxin Li, Aozhou Wang, Zengji Tu, Robert Tang, Yuxin Zhan, Eric Jiang, Yuxin Wu, Jianqing Zhang, Xiao Liang, Fang Wu, Haochi Zhang, Alexander Marlow, and Guancheng Wan. MetaRSI / RSI2: A Meta-Recursive Self-Improving System for Recursive Self-Improving Systems Themselves, 2026. URL https://arxiv.org/abs/2609.06396.

Tencent Cloud. CubeSandbox: Instant, Concurrent, Secure & Lightweight Sandbox for AI Agents. Software repository, 2026. URL https://github.com/TencentCloud/CubeSandbox. Accessed 6 September 2026.

Zefeng Wang, Minxi Yan, Jinhe Bi, Sikuan Yan, Volker Tresp, and Yunpu Ma. MetaSkill-Evolve: Recursive self-improvement of LLM agents via two-timescale meta-skill evolution, 2026. URL https://arxiv.org/abs/2607.05297.

Shujin Wu, Cheng Qian, Xiusi Chen, and Heng Ji. Teaching LLMs to self-evolve: Cultivating core meta-skills with reinforcement learning, 2026a. URL https://arxiv.org/abs/2607.21971.

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. ModularRSI: Modular and Generalizable Recursive Harness Self-Improvement, 2026b. URL https://arxiv.org/abs/2609.14857.

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. RRSI: Regularized Recursive Self-Improvement of Agent Harnesses, 2026. URL https://arxiv.org/abs/2609.24972.

Jundong Xu, Qingchuan Li, Jiaying Wu, Yihuai Lan, Shuyue Stella Li, Huichi Zhou, Bowen Jiang, Lei Wang, Jun Wang, Anh Tuan Luu, Caiming Xiong, Hae Won Park, Bryan Hooi, and Zhiyuan Hu. EvoArena: Tracking Memory Evolution for Robust LLM Agents in Dynamic Environments, 2026. URL https://arxiv.org/abs/2606.13681.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. OptiBench Meets ReSocratic: Measure and Improve LLMs for Optimization Modeling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=fsDZwS49uY.

Haoran Ye, Xuning He, Vincent Arak, Haonan Dong, and Guojie Song. Meta context engineering via agentic skill evolution, 2026. URL https://arxiv.org/abs/2601.21557.

Zhouliang Yu, Ruotian Peng, Keyi Ding, Yizhe Li, Zhongyuan Peng, Minghao Liu, Yifan Zhang, Zheng Yuan, Huajian Xin, Wenhao Huang, Yandong Wen, Ge Zhang, and Weiyang Liu. FormalMATH: Benchmarking Formal Mathematical Reasoning of Large Language Models, 2025. URL https://arxiv.org/abs/2505.02735.

Albert S. Yue, Lovish Madaan, Ted Moskovitz, DJ Strouse, and Aaditya K. Singh. HARP: A challenging human-annotated math reasoning benchmark, 2024. URL https://arxiv.org/abs/2412.08819.

Z.ai. GLM-5.3-Flash. Model card, 2026. URL https://huggingface.co/zai-org/GLM-5.3-Flash. Accessed 26 September 2026.

Guibin Zhang, Leo Lu, Fangzhou Xie, Kang Zhu, Junhao Wang, Zhifei Xie, Zhaochen Yu, Zihang Liu, Zhongxiang Sun, Qiankun Li, Yue Liao, Heng Chang, Xiaobin Hu, Qibing Ren, Wangchunshu Zhou, Chuanrui Hu, Yafeng Deng, and Shuicheng Yan. JIT-Agent: Scaling harness intelligence via just-in-time harness evolution, 2026a. URL https://arxiv.org/abs/2608.25593.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents, 2025a. URL https://arxiv.org/abs/2505.22954.

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

Jiayi Zhang, Yongfeng Gu, Jianhao Ruan, Maojia Song, Yiran Peng, Zhiguang Han, Jinyu Xiang, Zhitao Wang, Caiyin Yang, Yixi Ouyang, Bang Liu, Chenglin Wu, and Yuyu Luo. Harnessing agentic evolution, 2026c. URL https://arxiv.org/abs/2605.13821.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models, 2025b. URL https://arxiv.org/abs/2510.04618.

Wenting Zhao, Nan Jiang, Celine Lee, Justin T Chiu, Claire Cardie, Matthias Gallé, and Alexander M Rush. Commit0: Library Generation from Scratch, 2024. URL https://arxiv.org/abs/2412.01769.

Tailin Zhou. Hierarchical self-improvement: A framework for task-specific evolvable agent harnesses, 2026. URL https://arxiv.org/abs/2608.08466.

A TASK-POOL INVENTORY AND DATA SOURCES

The fixed manifests contain 62 source pools and seven target pools. A TaskPackage may represent a question, an interactive episode, a generated configuration, or a batch of examples; the counts below are local package counts, not upstream dataset sizes. Source pools provide 8,315 training and 1,592 validation packages and no final-test packages. Target pools provide 742 adaptation, 356 validation, and 1,078 final-test packages. The two roles therefore cover 69 pools and 12,083 packages.

Partition boundary. The source and target pool identifiers are disjoint. Within every pool, the manifest's train or adaptation, validation, and test task IDs are pairwise disjoint. Both METAAGENT training runs use the same source manifest. The target manifest is fixed for the held-out comparisons in Section 6. The holdout is by pool identifier, not necessarily by benchmark family; no target adaptation package enters METAAGENT training. We use the explicit allocations below rather than inferring a common split size from a launch-time fraction.

Data sources and local adaptations. The three classification target pools use the Meta-Harness data adapters (Lee et al., 2026b); LawBench also has its original benchmark source (Fei et al., 2023). The remaining target pools derive from Aider Polyglot (Aider contributors, n.d.), FormalMATH-Lite (Yu et al., 2025), OptMATH (Lu et al., 2025), and AgentBench DBBench (Liu et al., 2025). A shared source citation identifies the upstream project, not an assertion that the local TaskPackage adaptation reproduces the entire original benchmark or its official metric. Table 3 gives the full source-pool identifiers and cites the upstream projects for those with bibliography entries.

Table 3: Source-pool allocation in the current training manifest. Every listed pool has zero final-test packages.

Benchmark / source	Train	Val	Total
GPQA-HLE MCQA (Center for AI Safety et al., 2026; Rein et al., 2023)	48	32	80
HyperAgents SearchArena (Zhang et al., 2026b)	68	32	100
CodeI/O (Li et al., 2025b)	68	32	100
POLARIS Math (An et al., 2025)	68	32	100
HoVer-Hard (Jiang et al., 2020)	68	32	100
HyperAgents IMO (Zhang et al., 2026b)	268	32	300
Commit0 (Official) (Zhao et al., 2024)	34	12	46
DA-Code (Huang et al., 2024)	68	32	100
Compfiles (Official Proof) (Compfiles contributors, n.d.)	231	32	263
OlympiadBench (He et al., 2024)	68	32	100
Omni-MATH (Gao et al., 2024)	68	32	100
HARP (Yue et al., 2024)	37	13	50
Putnam-AXIOM (Gulati et al., 2025)	68	32	100
OptiBench (Static) (Yang et al., 2025)	68	32	100
IndustryOR (Huang et al., 2025)	68	32	100
Matbench (Dunn et al., 2020)	10	3	13
ORD Taxonomy (Kearnes et al., 2021)	57	32	89
ARC-AGI-1 (Official, 01) (Chollet, 2019)	168	32	200
ARC-AGI-1 (Official, 02) (Chollet, 2019)	168	32	200
ARC-AGI-1 (Official, 03) (Chollet, 2019)	168	32	200
ARC-AGI-1 (Official, 04) (Chollet, 2019)	168	32	200
ARC-AGI-2 (Official, 01) (Chollet et al., 2026)	248	32	280
ARC-AGI-2 (Official, 02) (Chollet et al., 2026)	248	32	280
ARC-AGI-2 (Official, 03) (Chollet et al., 2026)	248	32	280
ARC-AGI-2 (Official, 04) (Chollet et al., 2026)	248	32	280
Sokoban (Schrader, 2018)	45	15	60
Chess Puzzles (Lichess, n.d.)	45	15	60
Souper (Official-Derived) (Sasnauskas et al., 2018)	379	32	411
STOKE Hacker (Official-Derived) (Schkufza et al., 2012)	19	6	25

Continued on next page

Table 3 continued

Benchmark / source	Train	Val	Total
SyGuS Invariant (SyGuS Organization, n.d.)	397	32	429
SyGuS Nonconforming (SyGuS Organization, n.d.)	22	8	30
Large Text Compression (Mahoney, 2026)	45	15	60
Discuss EvoArena PersonaMem-Evo (Xu et al., 2026)	68	32	100
NL4Opt-E (Ramamonjison et al., 2022)	68	32	100
Reasoning Gym (Circuit Logic) (Stojanovski et al., 2025)	37	13	50
Reasoning Gym (Rubik's Cube) (Stojanovski et al., 2025)	37	13	50
Reasoning Gym (Tsumego) (Stojanovski et al., 2025)	37	13	50
Reasoning Gym (Mahjong Puzzle) (Stojanovski et al., 2025)	37	13	50
AlphaEvolve (Circle Packing) (Novikov et al., 2025)	68	32	100
AlphaEvolve (Unit Cube Packing) (Novikov et al., 2025)	68	32	100
AlphaEvolve (Heilbronn Fixed Triangle) (Novikov et al., 2025)	68	32	100
AlphaEvolve (Max-Min Distance Ratio) (Novikov et al., 2025)	68	32	100
Skill Self-Play BFCL V4 (Simple Live Simple) (Patil et al., 2025)	226	32	258
Skill Self-Play BFCL V4 (Simple Python) (Patil et al., 2025)	368	32	400
Skill Self-Play API-Bank (Level 1) (Li et al., 2023a)	367	32	399
Skill Self-Play API-Bank (Level 2) (Li et al., 2023a)	35	32	67
Skill Self-Play API-Bank (Level 3) (Li et al., 2023a)	99	32	131
Skill Self-Play ZebraLogic (Grid Small) (Lin et al., 2025)	128	32	160
ALFWorld Household (Shridhar et al., 2021)	468	32	500
FiNER-139 (Loukas et al., 2022)	30	10	40
AEGIS2 (Ghosh et al., 2025)	45	15	60
BIRD-Critic-1 (SQLite) (Li et al., 2025a)	84	32	116
MATH (Intermediate Algebra, Official) (Hendrycks et al., 2021b)	448	32	480
MATH (Number Theory, Official) (Hendrycks et al., 2021b)	448	32	480
MATH (Precalculus, Official) (Hendrycks et al., 2021b)	448	32	480
Formula-ACE (Han et al., 2024; Zhang et al., 2025b)	67	33	100
DROP (ADAS) (Dua et al., 2019)	90	10	100
MGSM (ADAS) (Shi et al., 2023)	90	10	100
IFBench (GEPA) (Pyatkin et al., 2025)	67	33	100
APPS (Python) (Hendrycks et al., 2021a)	90	10	100
TACO (Python) (Li et al., 2023b)	90	10	100
SearchQA (Author Snippets) (Dunn et al., 2017)	90	10	100
Total: 62 pools	**8,315**	**1,592**	**9,907**

Table 4: Target-pool allocations in the fixed held-out manifest. Adaptation packages are excluded from METAAGENT training.

Target pool	Adapt	Val	Test	Total
LawBench	200	50	100	350
Symptom2Disease	200	49	212	461
USPTO	50	30	100	180
Polyglot	47	12	166	225
DBBench	45	15	300	360
OptMATH	100	100	100	300
FormalMATH-Lite	100	100	100	300
Total: 7 pools	**742**	**356**	**1,078**	**2,176**

Interpretation of package counts. Some source pools contain generated instances or batched tasks. For example, the Matbench, FiNER, and AEGIS pools count local evaluation packages rather than individual upstream records (Dunn et al., 2020; Ghosh et al., 2025; Loukas et al., 2022). The BIRD-Critic SQLite pool is a training adaptation whose local score is a validity and partial-credit

proxy, not the official benchmark's semantic correctness score (Li et al., 2025a). We do not report source-pool scores as held-out benchmark results.

B Evolution Procedure and Implementation Details

Algorithms 2 and 3 expand the method overview in Algorithm 1. They specify the queue boundary, the paired node evaluation, and the evidence available to self-update. Appendix C gives the budgets used in the current experiments.

B.1 Global queue and node generation

Algorithm 2: MetaEvoForge training with a global node queue

Input: training pools $\mathcal{P}_{\text{train}}$, root package A_0, MetaAgent M_0
Require: node workers W, triple threshold K, update-cycle budget E
$\mathcal{C} \leftarrow \emptyset,\ Q \leftarrow \emptyset,\ M \leftarrow M_0,\ e \leftarrow 0$
initialize each T_P with A_0 and its validation score
while $e < E$ **do**
  if $|\mathcal{C}| \geq K$ **then**
    abandon outstanding jobs in Q; $Q \leftarrow \emptyset$ ▷ discard their later results
    $M \leftarrow \textsc{SelfUpdate}(M, \mathcal{C});\ e \leftarrow e + 1;\ \mathcal{C} \leftarrow \emptyset$
  end if
  if $e = E$ **then break**
  while $|Q| < W$ and $\textsc{Eligible}(\mathcal{P}_{\text{train}}) \neq \emptyset$ **do**
    $P \sim \textsc{Uniform}(\textsc{Eligible}(\mathcal{P}_{\text{train}}))$
    submit $\textsc{GenerateNode}(P, \textsc{SelectParent}(T_P), M)$ to Q
  end while
  if $Q = \emptyset$ **then break** ▷ no further node work is available
  $(P, v, \Delta) \leftarrow$ next completed job in Q; remove the job from Q
  if $v \neq \emptyset$ **then** $T_P \leftarrow T_P \cup \{v\};\ \mathcal{C} \leftarrow \mathcal{C} \cup \{\Delta\}$ **end if**
end while
return accepted M and the best validation-scoring package in each T_P

Algorithm 3: GenerateNode: one paired TaskAgent intervention

Input: pool P, parent u with package A_u and validation score S_u, MetaAgent M
Output: child node v and intervention Δ; or no child
$p \leftarrow \textsc{SampleTrainBatch}(P)$
$R_u \leftarrow \textsc{Evaluate}(A_u, p)$ ▷ training scores and traces
if R_u satisfies the configured saturation criterion **then return** $(\emptyset, \emptyset)$
$(\delta, A_v) \leftarrow \textsc{ProposeValidEdit}(M, A_u, p, R_u, S_u)$ ▷ bounded patch attempts
if no valid, nonempty package edit is obtained **then return** $(\emptyset, \emptyset)$
$R_v \leftarrow \textsc{Evaluate}(A_v, p)$
$S_v \leftarrow \textsc{ValidationScore}(A_v, P)$ ▷ scalar feedback; no validation traces
$v \leftarrow (A_v, u, S_v)$
$\Delta J \leftarrow J(A_v; p) - J(A_u; p)$
$\Delta \leftarrow (P, u, \delta, v, R_u, R_v, \Delta J, S_u, S_v)$
return (v, Δ)

Eligibility and collection. Only training pools supply node jobs and learning triples. At each dispatch, an eligible pool is sampled uniformly. A pool becomes ineligible when its node-attempt budget is exhausted or enough non-root nodes exceed the configured validation saturation threshold. Once the cache reaches K, the host abandons outstanding jobs for collection and excludes their later results using the collection-generation boundary. This is a logical cutoff; already running executions need not stop immediately. Node dispatch pauses during self-update. The cycle budget includes accepted, no-change, and rejected updates. If no further node work is available before the cache fills, remaining records are retained without an extra below-threshold update.

Parent selection and package versions. The reported training and held-out searches use validation-score-weighted parent sampling. Each tree retains package versions, parent links, and

edits. A child remains in the tree even when its score is lower than its parent's; final package selection uses validation score, with creation order breaking ties.

Bounded proposal attempts. A node samples and evaluates its parent training batch once. A saturated parent skips mutation. Proposal failures, invalid patches, and textually unchanged packages retry only patch generation, reusing the batch and parent evidence with compact feedback from the previous attempt. If the attempt budget is exhausted, no child or learning triple is produced. Successful package validation precedes child evaluation. The child training evaluation and its fixed validation evaluation can execute concurrently.

B.2 Evidence and MetaAgent self-update

Each consumed intervention identifies its pool, training tasks, parent and child package versions, and exact edit. It pairs parent and child execution traces and scores on the same sampled batch, together with their aggregate validation scores. The MetaAgent can inspect the training evidence and the edit, but a score change on one batch is not treated as evidence of a gain on every pool.

Self-update edits the MetaAgent's own allowed policy files, skills, memory, and executable components. The current instruction asks for a small, evidence-supported transferable correction before using no change as a fallback. TaskAgent proposals and self-updates cannot modify the shared harness engine, evolution code, evaluator, data splits, or permissions. Validation task content, answers, and traces are hidden; aggregate validation scores are visible during proposal generation and self-update. Final-test feedback is report-only.

Self-update is a bounded transaction. The last accepted MetaAgent is checkpointed; each candidate is checked for package validity and compliance with the edit boundary. A changed candidate must emit a valid patch on training-only proposal context. This check establishes proposal validity, not task improvement. Failed attempts restore the same accepted parent before retrying. The recorded outcomes are `accepted`, `no_change`, and `rolled_back`; only an accepted package change advances the successful MetaAgent version.

C Detailed Experimental Configuration

This appendix specifies the source training, held-out search, and model transfer protocols used for the current experiments. The package counts and pool identifiers are given in Appendix A.

C.1 Source training and checkpoint definition

The two MetaAgent training runs use the same 62 source-pool splits and seed 42. Their respective MetaAgent base models are `deepseek-v4-flash-0731` (DeepSeek-AI, 2026a) and `glm-5.3-flash` (Z.ai, 2026). Both use `Qwen3.6-35B-A3B` as the TaskAgent base model. Model weights remain fixed. The persistent MetaAgent package is the trained object, and the TaskAgent package is the object it edits within each pool.

Each run permits 32 MetaAgent self-update cycles. A cycle starts after $K = 24$ completed parent–child interventions have been collected. Thus "32 cycles" means 32 update attempts, not 32 passes over all training packages or 32 accepted package changes. Parent and child are each run once on the same batch of eight training packages. The batch is sampled without replacement within that intervention, but packages may recur in later interventions. Each child is also scored on its pool's fixed validation partition. Source-pool trees persist across cycles and permit at most 64 attempted child slots per pool. Parent selection uses variant C, which weights candidates by their validation scores. We retain the initial package M_0 and the packages after update cycles 8, 16, and 32.

At a collection boundary, outstanding node jobs are excluded from the update batch. A proposal may make at most two patch attempts; a self-update may make at most two attempts. A no-op proposal or saturated parent can consume an attempted slot without producing a child or intervention. Training also stops drawing from a pool after its node budget is exhausted or four non-root nodes exceed a validation score of 0.95. These rules mean that attempted slots, completed nodes, and collected

interventions are different counts. The asynchronous queue makes seed 42 insufficient for bitwise replay of the training trajectory.

C.2 HELD-OUT SEARCH AND FINAL SELECTION

For each checkpoint, search starts from the same root TASKAGENT package on each of seven target pools. The budget is 32 attempted child slots per pool. A skipped proposal still consumes a slot. The parent and child each run once on a matched sample of 16 adaptation packages. A generated child receives one evaluation on the pool's full fixed validation partition, which ranges from 12 to 100 packages across the seven pools. Variant C is also used for parent selection in this stage.

The highest-validation package is selected for final evaluation. Ties are resolved by creation order. Each final-test package is run once for that selected package; the root is evaluated directly as a separate control. Final-test scores never select a child. Diagnostic tests of additional candidates are excluded from the primary result table, even if one of them has a higher observed test score.

Within each model family, M_0 and the trained checkpoints use the same task IDs, root package, search budget, and execution settings on the seven target pools. These within-model comparisons therefore isolate checkpoint effects under a matched protocol, although independent search repeats are still needed to estimate variability. Absolute score comparisons between GLM and DeepSeek do not isolate training.

C.3 OUTCOMES AND RESOURCE ACCOUNTING

For pool P, final quality is the mean final-test score of the validation-selected package over that pool's fixed test packages. The seven-pool macro score is $\frac{1}{7}\sum_P S_P^{\text{test}}$; each pool has equal weight despite different test sizes. Search-efficiency curves track the best validation score seen so far against attempted child slots, cumulative METAAGENT tokens, and elapsed wall time. A validation curve is not a sequence of final-test scores. We separately count TASKAGENT executions, skipped or failed attempts, and METAAGENT training cost. Token and time comparisons require common accounting and shared endpoints before an efficiency claim can be made.

C.4 MODEL TRANSFER AND EXTERNAL METHODS

To test package portability across TASKAGENT models, we freeze the package selected by DeepSeek M_{32} while editing the Qwen TASKAGENT. We then run that unchanged package and its root with `deepseek-v4-flash-0731` as the TASKAGENT model on the same final-test task IDs. There is no new METAAGENT search or test-based selection in this direct-swap test. In a separate run, the same trained M_{32} METAAGENT starts from the same root and evolves a DeepSeek-driven TASKAGENT. We report the validation-selected package's final-test score on the same task IDs, with one test repeat. The original Qwen result provides a further cross-model reference, without a protocol-matched Qwen root. Comparisons across the independent runs are descriptive; a positive mean does not imply improvement on every pool.

Hyperagents, Meta-Harness, AHE, DGM, SIA-H, and RHO are candidate external comparisons. A method enters the seven-pool macro table only after its split, selection rule, search budget, score denominator, and execution status have been checked against this protocol. An absent or incomplete result remains unreported; it is not scored as zero.

C.5 CASE STUDY: REGRESSION IN AN EVOLVED AHE CANDIDATE

We observed a concrete candidate-level regression in the seed-42 AHE run on DBBench. The root agent scored 66.67% on validation. Across `node-001`, `node-002`, and `node-003`, training accuracy increased from 82.22% to 83.33% and 90.00%, whereas validation accuracy decreased from 73.33% to 66.67% and 60.00%. Thus, `node-003` performed better on the examples used to improve it but generalized worse than the root.

AHE created `node-003` after observing that the agent sometimes retrieved a value from the wrong database column. The new candidate added an automatic checker that inspected the selected column before submission. On one validation task, the question asked, "Which Attendance is the highest one

that has a Record of 15-28?" The agent correctly queried the `Attendance` column and initially obtained `32,502`.

The checker incorrectly warned that the question requested the `Opponent` column. Although the agent noted that this warning contradicted the question, the repeated warning made it revise the query and replace the correct answer, `32,502`, with `Baltimore Orioles`. The official evaluator marked the revised answer incorrect. In this case, a rule intended to prevent column-selection errors overrode correct task reasoning and introduced a new failure mode.

C.6 Execution environment and remaining controls

The runs execute TaskAgent packages in CubeSandbox (Tencent Cloud, 2026) using DeepSeek Harness (DeepSeek-AI, 2026b). Environments are resolved before task execution. The training configurations set temperature and top-p to 1.0, a per-request generation cap of 128,000 tokens, a TaskAgent iteration limit of 1,000, and 900-second TaskAgent and MetaAgent timeouts. Concurrency and model-service conditions should accompany wall-clock comparisons because they affect elapsed time without changing the attempted-slot budget.

D TRAINING DYNAMICS ACROSS METAAGENT BACKENDS

Figure 6 compares absolute best-so-far validation trajectories across the two runs. Figure 7 shows cumulative gain by pool, with pools ordered by mean final gain across the two runs.

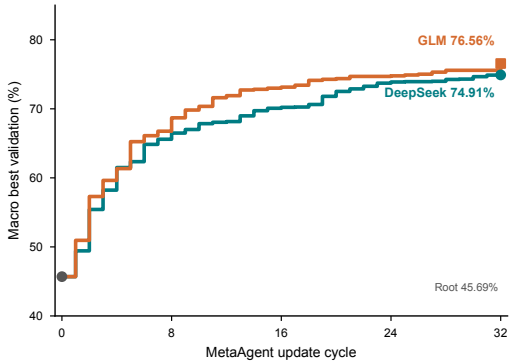


Figure 6: Macro-average best-so-far validation score across 62 training pools over 32 METAAGENT update cycles for DeepSeek and GLM.

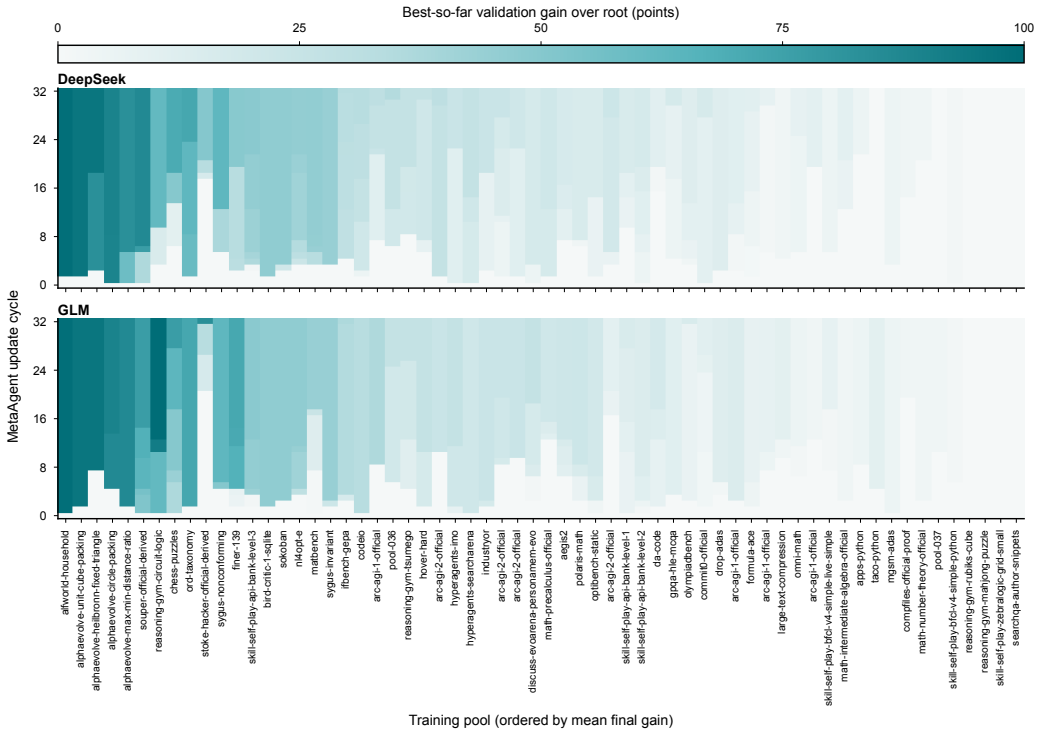


Figure 7: Per-pool cumulative best-so-far validation gain across 32 METAAGENT update cycles. Pools share a common ordering by mean final gain across the two runs.

E Comparison with Related Work

E.1 Unified Adaptation Protocol

Our goal is to preserve each method's original evolution mechanism while standardizing how methods interact with task environments. Each method still decides how to analyze trajectories, modify the TaskAgent, maintain candidates, and select a final version. The common evaluation layer requires only that the method export a frozen candidate TaskAgent package. The benchmark adapter runs this package in an isolated environment, invokes the task-specific scorer, and returns the score and shareable trajectory. Training examples and their feedback may be used to generate candidates; validation results are used only by the method's prescribed selection procedure; and the test partition is evaluated only after the final candidate is frozen.

Without a shared boundary, evaluating M meta-evolving methods on N benchmarks could require $M \times N$ pair-specific integrations. We instead implement one adapter per method that exports a frozen candidate through a common interface, and one adapter per benchmark that consumes this interface. This reduces the integration surface to $M + N$ adapters.

E.2 Meta-Evolving Method Adapters

Hyperagents. Hyperagents treats the task and meta programs as jointly evolvable state and selects parents from an archive of prior candidates (Zhang et al., 2026b). We preserve archive-based parent selection and joint inheritance: the selected parent MetaAgent actually generates the next candidate, while the paired TaskAgent is evaluated by the benchmark adapter. The original method can rewrite the Python programs implementing both agents. Our adaptation narrows this surface to two independent DeepSeek Harness (DSH) packages, whose persistent editable assets are skills, plugins, and profile configuration.

Meta-Harness. Meta-Harness searches over Python harnesses and uses persistent search history, execution feedback, and validation performance to maintain a candidate frontier (Lee et al., 2026b). On the text-classification benchmarks, we retain its native candidate interface: the generated Python program controls context construction and answer production. On environment-based tasks, the benchmark runtime and scorer cannot be replaced by arbitrary candidate code. There, a Meta-Harness candidate acts as a generator that constructs per-task guidance, expressed as a prompt or skill, for a clean DSH TaskAgent. The evolving asset is therefore the program that generates this guidance rather than the protected environment executor.

AHE. AHE diagnoses failures from TaskAgent trajectories and uses the resulting evidence to revise an agent harness (Lin et al., 2026). We preserve its evaluate–diagnose–summarize–modify–reevaluate loop and its attribution of later behavior to specific edits. The original method can edit most components of a NexAU workspace, including prompts, skills, tools, middleware, memory, and agent-loop code. We map these edits to a DSH package's system prompt, skills, plugins, and profile.

DGM. The Darwin Gödel Machine (DGM) lets a coding agent modify its own Python implementation and retains successful candidates as possible parents for continued search (Zhang et al., 2025a). We preserve self-modification, archive maintenance, and parent selection, but replace the editable coding-agent repository with a DSH TaskAgent package.

SIA-H. SIA-H is our harness-only adaptation of SIA (Hebbar et al., 2026). A fixed Feedback Agent reads the current TaskAgent's training trajectories and scores, proposes a patch, and continues along a linear inheritance chain. Full SIA can rewrite a task-specific Python scaffold and update model weights. Adapted SIA-H does neither: its persistent candidate is a DSH package, and patches are restricted to skills, plugins, and profile configuration.

RHO. Retrospective Harness Optimization (RHO) first probes the root agent, selects a representative subset of training tasks, repeatedly evaluates candidates, and uses diagnosis, multi-candidate generation, and self-preference comparisons to decide whether to accept a new harness (Pan et al.,

2026). We preserve task probing, subset selection, repeated rollouts, diagnosis, candidate deduplication, and the method's self-preference acceptance rule. Its original persistent instructions, skills, tools, memory, and helper scripts are mapped to the system prompt, skills, plugins, and profile of a DSH package.

E.3 Benchmark Adapters

LawBench. We use the criminal-behavior and charge-prediction subtask from LawBench (Fei et al., 2023). Each case becomes an independent classification task: the TaskAgent reads the case description and submits a normalized charge label, which is compared with the reference label outside the agent environment. All methods use the same frozen adaptation, validation, and test members. Validation and test labels remain hidden from both the TaskAgent and the improver. The reported result therefore concerns this subtask rather than the full LawBench suite.

Symptom2Disease. We use the Symptom2Disease classification adapter released with Meta-Harness (Lee et al., 2026b). The TaskAgent maps a symptom description to one label from the benchmark's disease set, and the host-side scorer applies label normalization followed by exact matching. We retain the source text and label set, freeze a common split for all methods, and remove one validation example that duplicated a test example. Validation and test labels are never placed in the candidate workspace.

USPTO. We use the USPTO adapter and scoring semantics from Meta-Harness (Lee et al., 2026b). Given a target product, the TaskAgent predicts the reactants and submits canonicalized SMILES. Host-side code performs normalization and correctness checking. All methods use the same frozen data version and answer interface; this pool is not mixed with other USPTO-derived tasks in the repository.

Polyglot. Polyglot is a multilingual code-repair benchmark (Aider contributors, n.d.). Each task supplies a project that the TaskAgent can inspect and modify in an isolated workspace. After the run, the benchmark adapter invokes the fixed tests and scores whether they all pass. Language runtimes, dependencies, time limits, and tests are managed by the benchmark adapter, not by the evolving method.

DBBench. DBBench originates from AgentBench and evaluates whether an agent can translate a natural-language request into correct database operations (Liu et al., 2025). Our adapter asks the TaskAgent to produce an ordered action sequence and deterministically replays those actions in a fixed database environment before scoring their effect. This preserves the task and execution-based scoring semantics but does not recreate the original online AgentBench dialogue session; we treat that difference as part of the adapted evaluation.

OptMATH. OptMATH requires the TaskAgent to turn a natural-language optimization problem into a complete executable mathematical program (Lu et al., 2025). The candidate runs in an isolated environment with fixed Python, Gurobi, and supporting dependencies. The benchmark adapter executes the generated program and checks its solution rather than comparing a free-form textual answer. Problem statements, solver access, dependencies, and execution limits are shared across methods.

FormalMATH-Lite. FormalMATH-Lite is the official subset of FormalMATH (Yu et al., 2025). The TaskAgent receives a theorem and its required context and submits a Lean proof. The benchmark adapter compiles the candidate in a fixed Lean environment and assigns credit only when the proof checker accepts it. Lean dependencies and the checker remain outside the editable candidate package, and reference proofs are never exposed to either the TaskAgent or the improver.

F Responsible-Use Statement

Automating agent improvement could reduce development costs and make useful tools more accessible. It could also propagate faulty or misaligned behavior across many agents, amplify harmful

capabilities, and concentrate control in the systems that define objectives and evaluators. Repeated optimization may favor benchmark-specific shortcuts over reliable behavior and consume substantial computational resources. Our experiments concern bounded benchmark tasks; their scores do not establish safe deployment in settings that affect people.

The evaluated workflow uses scoped package edits, versioned parent–child records, held-out tests, and isolated execution. These measures aid inspection and containment but do not guarantee safety. For deployment, we recommend least-privilege tools, resource limits, independent testing for misuse and distribution shift, and human review before promoting evolved agents. Operators should retain rollback and halt controls outside the editable agent package, protect sensitive data and credentials, and assign clear responsibility for approving changes. Agents that manage people or act in high-impact domains require additional domain-specific assessment and accountable human oversight.