ASPIRE: CAN MODELS SELF-EVOLVE FROM VAGUE GOALS?

Anonymous authors
Paper under double-blind review

ABSTRACT

Many important forms of human learning begin with a vague goal, such as "become a better physicist" or "improve at research." Learners must interpret the goal, identify capability gaps, decide how to learn, and determine whether they have actually improved. In contrast, existing work on LLM self-evolution typically begins with tasks and evaluation metrics specified by humans, reducing self-evolution to optimizing an explicit objective rather than deciding what and how to learn. We introduce ASPIRE, a benchmark for vague-goal-driven self-evolution. ASPIRE provides only a natural-language capability goal while downstream evaluation tasks remain hidden. The agent must operationalize the goal by choosing data and update methods, constructing training and validation signals, and deciding when to evaluate. ASPIRE supports both model-weight and agent-harness evolution in a unified interactive environment and evaluates the resulting systems on a hidden, expert-authored set of 520 items spanning six goals. Our experiments show that vague goals redirect search effort toward goal interpretation. Current agents routinely complete training and harness-editing loops, but weight-level gains remain sparse and unstable, and the strongest evolved harness remains below the engineered Qwen-Agent reference. Agents often train on mismatched data and trust narrow self-evaluations, so local gains fail to transfer to hidden evaluation and continued search and training can erase earlier improvements.

1 INTRODUCTION

Many important forms of human learning begin with a broad capability direction rather than a predefined benchmark, training set, or fully computable reward function. A student seeking to become a better physicist, for example, not only chooses textbooks, exercises, and study methods, but also identifies gaps in their knowledge, prioritizes which capabilities to develop, and determines whether learning has produced genuine progress. Autonomous learning therefore involves three coupled decisions: *what* to improve, *how* to improve it, and *how to verify* the improvement.

Existing work on LLM self-evolution focuses primarily on the second decision. Given a concrete task, evaluation script, and success metric, LLM agents can already collect data, run training, and revise post-training strategies from feedback. PostTrainBench (Rank et al., 2026), LaMDAgent (Yano et al., 2025), Evo-Memory (Wei et al., 2026), and SEAL (Zweiger et al., 2025) show that modern LLMs can autonomously search for effective optimization paths toward a specified objective. These systems therefore begin after humans have operationalized a broad capability request such as "improve mathematical reasoning" into a fixed task-level objective such as "improve performance on AIME" (Figure 1(a)). The task format, difficulty range, metric, and success criterion are already defined: the agent searches over *how* to improve, but not *what* capability objective to pursue. When existing benchmarks saturate or cease to reflect emerging capability gaps, humans must still discover new weaknesses, translate them into tasks, and design corresponding benchmarks and rewards. This leaves a central question for sustained recursive self-improvement:

> **Given only a vague goal, without an agent-visible task specification or decomposable reward, can a model determine what it should learn and how to improve itself?**

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

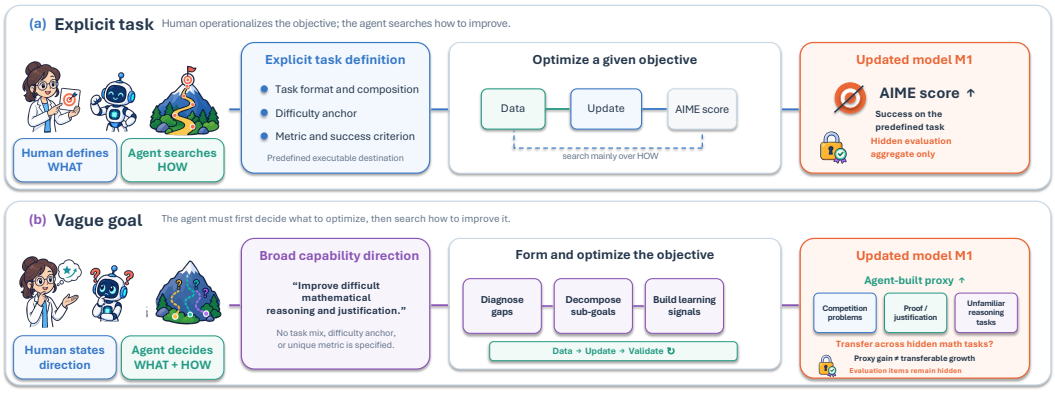

Figure 1: From explicit-task optimization to vague-goal-driven self-evolution. (a) With an explicit task, humans fix the task, metric, and success criterion; the agent searches over *how* to improve. (b) With only a broad capability direction, the agent must also decide *what* to optimize and build its own learning and validation signals. Sealed external evaluation tests whether proxy gains transfer.

We formalize this problem as *vague-goal-driven self-evolution*. As illustrated in Figure 1(b), "vague" does not mean ambiguous or poorly specified. It denotes a broad capability direction that has not yet been operationalized as a fixed task-level objective and evaluation metric. The agent must therefore diagnose capability gaps, form intermediate objectives, construct learning and validation signals, and jointly search over *what* to optimize and *how* to optimize it. This expanded decision space introduces a deeper failure mode: gains on an agent-constructed proxy may not translate into genuine improvements in the intended capability. Evaluating vague-goal-driven self-evolution therefore requires a goal-aligned external evaluator whose task-level specification and items remain hidden from the agent.

To study this problem, we introduce ASPIRE, a benchmark for vague-goal-driven self-evolution. ASPIRE provides only a natural-language capability goal while downstream evaluation tasks remain hidden. The agent decides how to operationalize the goal, including what data and update methods to use and whether and how to perform intermediate validation. ASPIRE supports evolution at two levels: model weights and the supporting agent harness. A unified agent-facing tool allows the agent to search for, download, import, or synthesize data; launch different forms of training; run self-evaluations; and manage candidate branches and selection. The controller handles data management, job scheduling, resource isolation, and checkpoint verification. ASPIRE records the resulting decisions, actions, and self-evaluation trajectories, and evaluates the resulting systems on a hidden, expert-authored set of 520 items spanning six goals. Depending on the protocol, the agent receives either no intermediate evaluation score or only bounded aggregate scores; it never accesses evaluation items, reference answers, or per-item feedback.

We organize our experiments as a progression from controlled comparison to autonomous evolution. RQ1 isolates the effect of goal specification by replacing an explicit post-training task with a vague capability goal under matched starting models, update tools, and compute budgets. RQ2 then moves to the full ASPIRE setting and asks whether an agent starting from an instruction-tuned checkpoint can convert a vague goal into a retained weight-level capability gain. RQ3 holds model weights fixed and shifts the object of evolution to the supporting agent harness.

RQ1: What changes when an explicit post-training task is replaced by a vague capability goal? Vague goals redirect search effort toward goal interpretation and operationalization and, in the evaluated settings, yield lower aggregate outcomes than the corresponding explicit-task references.

RQ2: Starting from an instruction-tuned checkpoint, can an LLM turn a vague goal into a retained capability gain through self-directed weight updates? Agents routinely complete data selection, training, and checkpoint generation, but above-base improvements on hidden evaluation data remain rare and are not reliably retained through continued search.

RQ3: Beyond model weights, can the agent harness itself evolve? With model weights fixed, agents can generate functional successor harnesses for goal interpretation, tool use, and self-evaluation, but even the strongest observed successor remains below the engineered Qwen-Agent reference (Qwen Team, 2023).

Our contributions are threefold. First, we formalize target operationalization—the conversion of a broad capability goal into trainable objectives, learning signals, and validation criteria—as a missing axis of autonomous post-training. Second, we introduce a benchmark with sealed evaluation and a minimal interactive environment for both model-weight and harness updates. Third, outcome-and-trajectory evidence reveals the gap between executing updates and retaining target-aligned gains.

2 ASPIRE: SELF-EVOLUTION BENCHMARK AND INTERACTIVE ENVIRONMENT

2.1 TASK DEFINITION AND EVOLUTION SURFACES

Campaign and round. Let $G \in \mathcal{G}$ denote a *vague goal*: a natural-language capability objective that does not specify a training task, dataset, or optimization procedure. Unlike an explicit-task setting, the agent must perform *goal operationalization* by translating G into its own data, update objective, and validation criteria. A campaign fixes the controller-side experiment contract

$$\Gamma = (G, \mathcal{E}_G, J, \mathcal{A}, B, \Sigma), \qquad Y_r^{\text{in}} = (M_r, H_r, D_r),$$

where $\mathcal{E}_G$ is the versioned evaluator bound to G, J is its judge when model-based scoring is required, $\mathcal{A}$ is the typed action contract, B is the campaign budget, and Σ is the predeclared terminal selection rule. The incoming state contains model weights M_r, an agent harness H_r (the runtime instructions, tool policy, workflow, memory, and validation logic around the model), and the decision model D_r that directs the search. The evaluator—including its items, answers, rubrics, routing metadata, and scoring configuration—exists throughout the campaign but is never part of the agent's observation. RQ1 is the vague-goal counterpart of PostTrainBench: it preserves each original sealed task evaluator while replacing the explicit benchmark identifier with a broad capability description. These task-specific results remain separate from, and are not pooled with, the six-goal ASPIRE evaluation used in RQ2 and RQ3.

Every weight candidate records its parent checkpoint, registered data, update specification, and provenance for the agent's own validation data. Every harness candidate records its parent version, content hash, and edit provenance. We use *Self* for the configuration in which the frozen base checkpoint acts as the decision model D_0, while candidate updates descend from the same checkpoint M_0. Promotion of a trained descendant is an explicit between-round operation and never an automatic within-round replacement. Additional campaign and round details appear in Appendix E.

2.2 A HIDDEN, QUERY-LIMITED EVALUATION SET

Measurement without an agent-visible benchmark. Learning from a vague goal cannot be evaluated without an external criterion of progress. ASPIRE therefore retains a benchmark on the controller side while hiding its task-level specification from the agent. The hidden evaluation set is scientific instrumentation, not an agent-visible learning contract: it lets researchers measure whether an agent-selected update improves the intended capability without disclosing which tasks define success. It contains 520 expert-authored evaluation items covering six vague goals: scientific and academic reasoning (75), humanities and social-science knowledge (110), health and medical reasoning (100), mathematical reasoning (126), logic, reliability, and instruction following (89), and academic and scientific writing (20). Each top-level item belongs to exactly one goal; the reliability goal is composite, and writing items are task bundles with potentially multiple responses. Each goal is evaluated on its corresponding items, which the agent never sees, and the protocol returns no item-level result—only an aggregate score when feedback is permitted.

Information boundary and interpretation. The agent receives a vague goal expressed in natural language and may construct its own training data and its own validation data. It cannot access hidden evaluation items, reference answers, routing labels, rubrics, candidate outputs, or judge traces. Every dataset registered for training is checked for overlap with the hidden evaluation set before admission

to the training backend. Evaluation returns only the aggregate score permitted by the protocol and the remaining query allowance under a campaign-fixed evaluator and judge configuration.

Under the adaptive-feedback protocol, a bounded number of queries may evaluate the same items for a goal. The resulting aggregate scores form a sparse black-box outcome channel: the agent may use them to choose subsequent updates, branches, or a stopping point, but does not observe the items and cannot train on their contents. This protocol therefore measures performance under bounded adaptive feedback rather than on an untouched post-selection test. Under the final-only protocol, the agent may train multiple checkpoints but receives no evaluation score before submitting its single terminal checkpoint, so that score cannot guide further updates. In RQ3, the harness candidate is frozen before its first evaluation, and the resulting score is not returned to its creator.

2.3 A Minimal Interactive Environment with Safe Retention

ASPIRE retains controlled compute and real model updates while deliberately separating strategic and infrastructural complexity. The agent controls how to operationalize the given goal and how to update the model or harness; the controller absorbs credentials, storage conventions, distributed-job mechanics, and recovery. This keeps the interface simple without prescribing the learning strategy.

Minimal actions and trust boundary. For weight evolution, the agent chooses data, update method, hyperparameters, parent checkpoint, and whether to continue, branch, or stop. For harness evolution, it edits the runtime prompt, tool policy, workflow, memory, or validation procedure using its own data and freezes a versioned candidate. The interface exposes only semantic actions—registration, training or editing, verification, validation, evaluation, branching, and termination—while hiding backend mechanics. It preserves the information boundary defined in Section 2.2: the agent makes strategic decisions, the controller executes and constrains them, and evaluation items remain hidden.

Candidate outcomes and safe retention. Let Y^{in} be the incoming state and $\mathcal{K}$ the verified candidates produced in the round. For candidate k, we define the raw score change as

$$\Delta^{\mathrm{raw}}(k) = \widehat{s}_\Gamma(Y(k)) - \widehat{s}_\Gamma(Y^{\mathrm{in}}),$$

which may be positive or negative. The *best evaluated checkpoint* is the highest-scoring evaluated checkpoint; the *selected checkpoint* is the highest-scoring eligible checkpoint under the campaign's predeclared eligibility and completion rules. For score-gated weight evolution, the controller retains the selected checkpoint $k^\star$ only when $k^\star \neq \perp$ and $\Delta^{\mathrm{raw}}(k^\star) > 0$; otherwise it rolls back to Y^{in}. The resulting state Y^{out} is therefore $Y(k^\star)$ in the former case and Y^{in} otherwise. Consequently,

$$\Delta^{\mathrm{ret}} = \widehat{s}_\Gamma(Y^{\mathrm{out}}) - \widehat{s}_\Gamma(Y^{\mathrm{in}}) \geq 0.$$

The nonnegative retained improvement follows from safety rollback and is therefore a selection outcome, not evidence that every attempted update improves capability. Harness candidates are instead compared post-freeze with model weights fixed; their effect may be negative and does not imply deployment or recursive handoff. Full eligibility, tie-breaking, termination, and aggregation rules for the reported experiments appear in Appendix C.1. The candidate verification rules appear in Appendix E.3; the execution interface is shown in Figure 7.

3 Experiments and Analysis

We organize the experiments around the three questions introduced in Section 1. **RQ1** asks how vague goals change post-training outcomes and search trajectories relative to an explicit-task setting. **RQ2** asks whether an LLM can turn a vague goal into a capability gain through weight updates. **RQ3** asks whether the agent harness can itself evolve when runtime model weights are fixed. Together, the three studies move from characterizing the effect of goal specification, to evolving the model, and finally to evolving the system around it.

3.1 Setup

RQ1 changes goal specification and uses the original PostTrainBench evaluator. RQ2 uses two complementary protocols over six vague goals: the final-only protocol comprises 24 runs without

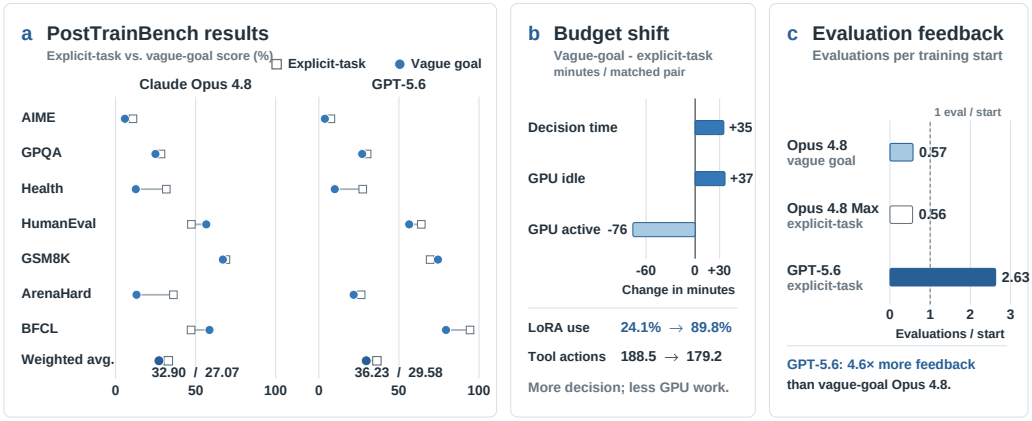

Figure 2: RQ1: vague-goal versus explicit-task post-training. (a) Final scores on seven PostTrainBench tasks; explicit-task values are official references. (b) In matched Claude Opus 4.8 pairs, vague goals add decision and GPU-idle time and reduce active GPU time. (c) Evaluations per training start (descriptive cross-system comparison). Exact values appear in Appendix B.

intermediate evaluation scores, with two terminal runs per model–goal pair reported as their arithmetic mean (Avg@2); the adaptive-feedback protocol comprises 30 configuration–goal cells and tests feedback-guided checkpoint search. RQ3 changes the harness while fixing the runtime model and hidden evaluation set.

We report the two RQ2 protocols separately. In RQ3, three executions of each frozen harness measure runtime variation, not three independent evolution runs. Detailed budgets, evaluator settings, and outcome definitions appear in Appendix C.

3.2 RQ1: How Vague Goals Change Post-Training

Setting. Standard PostTrainBench names the downstream benchmark, thereby specifying what the agent should optimize. We replace that identifier with a broad capability description while preserving the post-training interface and original task evaluator. The resulting agent must choose, explicitly or implicitly, a concrete goal, a proxy training task, and a validation method in addition to the update itself. We use the released PostTrainBench scores as a reference and matched Claude Opus 4.8 trajectories under explicit-task and vague-goal contracts only for process comparisons. Other run metadata can also differ, so the results below describe the observed difference rather than isolate a prompt-only causal effect. Tasks, models, and pairing criteria are given in Appendix B.

Main result: vague-goal final scores are lower overall, but task-level changes vary. Figure 2(a) shows that Claude Opus 4.8 under vague-goal prompting reaches 27.07, compared with the 32.90 official Claude Opus 4.8 Max reference from PostTrainBench; GPT-5.6 reaches 29.58, compared with the 36.23 official GPT-5.6 reference. Yet the task-level pattern varies: Claude Opus 4.8 under vague-goal prompting is higher on HumanEval and BFCL, and GPT-5.6 under vague-goal prompting is higher on GSM8K. We therefore interpret the aggregate gaps without assuming uniform task-level degradation.

Trajectory pattern: vague goals are associated with different resource use. In the matched Claude Opus 4.8 runs, vague-goal trajectories record 2,109 more seconds of decision-model thinking and 0.61 more GPU-idle hours per matched run pair, while active training and evaluation time falls by 1.27 hours (Figure 2(b)). Accesses to agent-visible task materials and proxy evaluation scripts become 2.98× and 2.39× as dense. In a separate trace-level sample spanning the six shared settings, the mean number of tool actions does not increase (188.5 to 179.2), while LoRA is used more often under vague goals. These descriptive shifts are consistent with more goal-operationalization work and less active update time, but do not establish that one displaces the other or identify which choice causes the final score gap. Additional RQ1 reference comparisons appear in Appendix F.1.

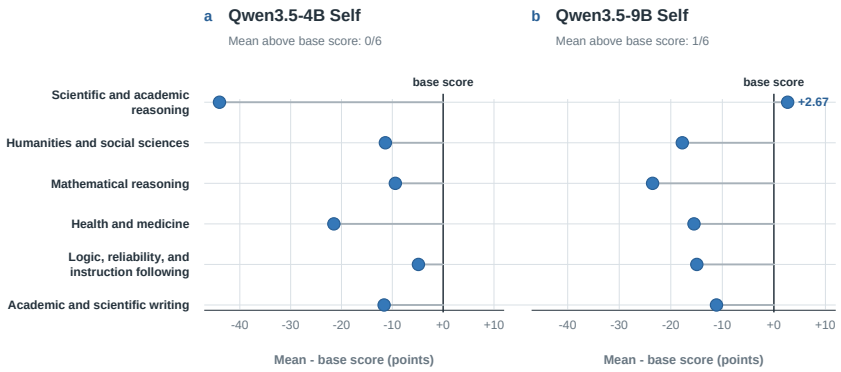

Figure 3: Final-only protocol: change from the base score before rollback, averaged over two runs per model–goal pair. Only Qwen3.5-9B on scientific and academic reasoning exceeds its base score; run-level scores appear in Appendix C.3.

3.3 RQ2: Self-Directed Weight Evolution from Vague Goals

Settings. Unlike RQ1, which adapts pretrained models, both RQ2 settings begin from instruction-tuned checkpoints and allow iterative training. We call each initial instruction-tuned checkpoint the *base model* and its initial evaluation score the *base score*. This is a practically important setting: an update should add goal-relevant behavior without disrupting useful behavior already encoded by instruction tuning. The final-only protocol tests open-loop execution, whereas the adaptive-feedback protocol tests search with sparse aggregate scores. Although the system can promote a trained descendant to the decision model in a subsequent round, all reported runs keep the initial decision model fixed; the evidence therefore concerns adaptive weight search rather than recursive decision-model replacement. In both protocols, evaluation focuses on one goal and does not measure preservation of every unrelated capability.

Final-only protocol. Qwen3.5-4B Self and Qwen3.5-9B Self (Qwen Team, 2026) each optimize all six goals in two separately executed terminal runs, yielding 24 runs in total. For each model–goal pair, we report Avg@2—the arithmetic mean of the two final-checkpoint scores—rather than best-of-two. A run may launch multiple training jobs within 40 GPU-hours, but it may submit only one final checkpoint for evaluation on the hidden evaluation set. The two-run repetition makes the final estimate less dependent on any single run. This protocol does not isolate decision-model scale because the decision model, the base model, and the base score change together between 4B and 9B.

Final-only result: only one two-run mean exceeds the base score. All 24 final-only runs complete their single final-checkpoint evaluation (Figure 3). Taking each model–goal pair as the reporting unit, Qwen3.5-4B has 0/6 mean outcomes above its base score and Qwen3.5-9B has 1/6. The sole positive pair is scientific and academic reasoning: both runs rise from 45.33 to 48.00, so the two-run mean is 48.00, or 2.67 points above the base score. At the pair level, this is the only positive mean among all 12 model–goal pairs. At the individual-run level, 3/24 final checkpoints exceed their base scores, while rollback retains the base model in the other 21 runs. Individual terminal-run and per-goal results appear in Appendix C.3.

Adaptive-feedback protocol. The adaptive-feedback protocol asks whether sparse aggregate evaluation scores can guide an improving multi-checkpoint trajectory. The decision model chooses public data, an update method, a training plan, and when to evaluate or branch in response to the scores observed so far. In the Self settings, the initial Qwen3.5-4B or Qwen3.5-9B checkpoint remains the fixed decision model while descendant checkpoints are trained. GPT-5.6 Luna, Terra, and Sol serve as external decision models for the same Qwen3.5-4B base model. Eligible checkpoints may receive repeated aggregate scores on the hidden evaluation items for a goal, but the agent never sees the items or per-item feedback.

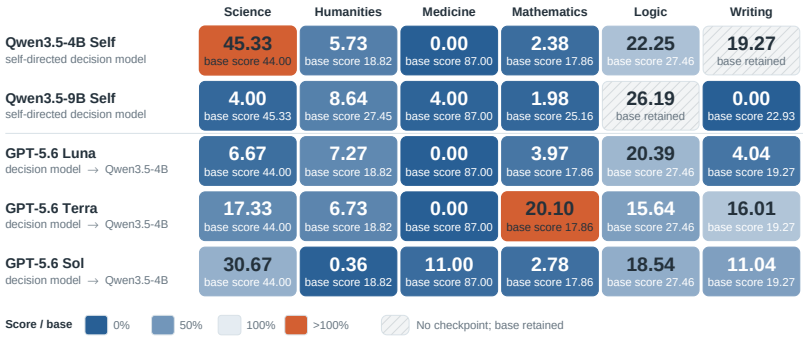

Figure 4: Adaptive-feedback protocol: best evaluated checkpoint score (large) and base score (small) for all 30 cells. Fill shows the ratio to base (orange: above base; hatched: no evaluated checkpoint).

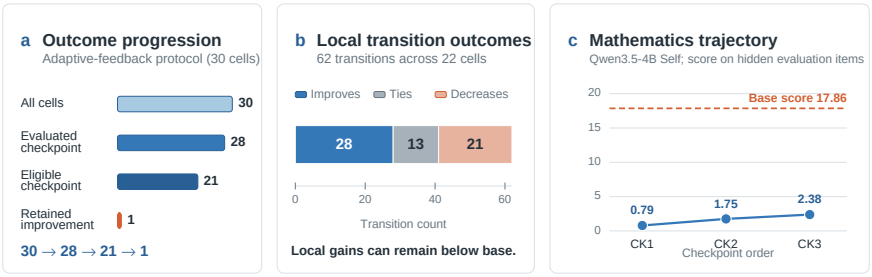

Figure 5: Adaptive-feedback outcomes. (a) 28 of 30 cells yield an evaluated checkpoint, 21 an eligible one, and 1 a retained gain. (b) 28 of 62 checkpoint transitions increase. (c) A rising trajectory can stay far below base.

Adaptive-search result: producing a trained checkpoint is common; producing an above-base goal score is rare. Across the 30 configuration–goal cells, with one run per configuration–goal cell, 28 produce an evaluated checkpoint and 21 satisfy the predeclared provenance, exploration, completion, and terminal-record requirements for an eligible checkpoint (Figures 4 and 5(a)). Only two cells have a best evaluated checkpoint whose score exceeds the base score. Qwen3.5-4B Self raises science from 44.00 to 45.33, but that run does not satisfy the complete-run requirement, so rollback keeps the base model and no Self cell retains a model above its base score; Terra raises Qwen3.5-4B mathematics from 17.86 to 20.10 and is the only cell whose checkpoint scores above the base score, passes eligibility, and remains after rollback. Thus the agents often close the executable training loop—data, update, checkpoint, and evaluation—but most resulting checkpoints do not preserve the base score, and several collapse to near zero. The 45.33 science score is a best evaluated checkpoint from one adaptive-feedback run and is not reproduced by the two final-only Qwen3.5-4B final checkpoints. Because feedback and checkpoint-selection contracts differ, we treat the protocols as complementary settings rather than repeated trials. The retained Terra score was selected using repeated aggregate feedback on the same fixed evaluation slice; without a separate confirmation slice, it is evidence of feedback-guided selection rather than an independently replicated capability gain.

Trajectory pattern 1: apparent progress can be recovery from training-induced regression. The 22 cells with multiple evaluated checkpoints contain 62 consecutive transitions: 28 rise, 13 tie, and 21 fall. Later search beats the first evaluated checkpoint in 14/22 cells, but this does not imply a score above the base score. For example, Qwen3.5-4B Self improves on mathematics from 0.79 to 1.75 to 2.38 while remaining far below its 17.86 base score (Figure 5(c)). A positive within-lineage slope can therefore describe recovery from training-induced regression rather than an above-base improvement. The transition counts are pooled because cells contribute different numbers of checkpoints.

Observed failure mode: numeric-label SFT is associated with answer-format collapse. The Self trajectories show concentration in both domain and answer format. In the adaptive-feedback pro-

tocol, Qwen3.5-4B Self uses GSM8K or Hendrycks mathematics data in 30/32 dataset-import events, including runs targeting science, logic, and writing. In the final-only protocol, five submitted checkpoints are trained on numeric-label MMLU SFT: all 21,000 training targets are single-digit labels, and all 279 corresponding evaluation outputs are single digits. Their scores are 0, 0, 0, 6.141, and 0. All five checkpoints show the same pattern. This is not a randomized causal comparison, but it documents a concrete answer-format mismatch associated with training-induced regression. Because all 24 final evaluations complete without item errors, missing outputs or evaluator failures do not explain the aggregate degradation. Further accounting appears in Appendix C.3; content-dependent interpretations use the controlled-access source traces. Additional checkpoint trajectories appear in Appendix F.2.

3.4 RQ3: Harness Evolution

Design. We fix Qwen3.5-4B as the runtime model and use the original Qwen-Agent (Qwen Team, 2023) as the reference harness. Under the same editable contract, Qwen3.5-4B Creator and GPT-5.6 Luna, Terra, and Sol each attempt to produce one successor harness H_1 from the same H_0. The Qwen3.5-4B Creator produces no valid H_1 and is assigned zero by the predeclared failure rule. Each of the three valid successors is frozen and executed three times on the same 20 hidden evaluation items for the academic and scientific writing goal; those scores are not returned to the creator. There is one creation trajectory per condition, while the three executions measure runtime variation of a frozen harness. No run evolves H_1 into H_2, so the experiment evaluates one-step harness generation rather than recursive harness self-improvement.

Result: the closest successor harness is numerically below the fixed reference. All three valid successors are numerically below the original Qwen-Agent, which obtains 28.64 task macro and 27.65 example micro (Table 1). The Sol-created successor is closest at 27.22/25.97, 1.42 task-macro points below the reference, followed by the Terra-created successor at 20.76/20.14 and the Luna-created successor at 19.32/18.33. Creator-trajectory analysis and the research-question summaries appear in Appendix F.

Table 1: RQ3: hidden-set scores of the fixed Qwen-Agent reference and one-step successor harnesses, all with Qwen3.5-4B as the runtime model.

Configuration	Task macro	Example micro
Qwen-Agent reference	28.64	27.65
Qwen3.5-4B Creator (no valid harness)	N/A	N/A
GPT-5.6 Luna successor	19.32	18.33
GPT-5.6 Terra successor	20.76	20.14
GPT-5.6 Sol successor	27.22	25.97

Note: Means over three executions of each frozen harness on the 20 writing items. Task macro weights items equally; example micro weights scored examples equally. The Creator yields no valid harness (failure score zero). Dispersion is not reported, so differences are descriptive.

4 Related Work

Self-training and self-evolution methods. Self-evolution methods let models construct parts of their own learning signal. STaR (Zelikman et al., 2022) fine-tunes on self-generated rationales that yield correct answers; ReST-EM (Singh et al., 2024) iterates sampling, reward filtering, and retraining; Self-Rewarding LM (Yuan et al., 2024) uses the model as judge; and SPIN (Chen et al., 2024) learns through self-play. R-Zero (Huang et al., 2025) and Absolute Zero Reasoner (Zhao et al., 2025) further reduce dependence on external data by posing, solving, and refining problems, while SEAL (Zweiger et al., 2025) reinforces model-generated self-edits according to downstream gains. In all cases, the model helps generate data, rewards, tasks, or edits, but progress is still operationalized by a concrete signal such as correctness, reward, self-play outcome, or task gain. ASPIRE is complementary: it introduces no new training algorithm, but tests whether a model given only a vague goal can construct an effective learning process around it.

Automated post-training and ML-agent benchmarks. PostTrainBench (Rank et al., 2026) gives an agent a base model, evaluation script, and budget for data filtering, training, and hyperparameter search; LaMDAgent (Yano et al., 2025) similarly automates iterative post-training pipelines. RSIBench-Data (Meng et al., 2026) fixes the target model, benchmark, and training stack while agents revise data strategies, and finds that most continued searches finish below their own best checkpoint—a retention failure related to our distinction between candidate progress and retained gain. AI4AI-Bench (Chi et al., 2026) instead lets agents rewrite a training algorithm under hidden evaluation, yet most submissions do not change how the model learns. PAST-Bench (Xue et al., 2026) tests whether retained experience improves later personal-agent tasks through the save–retrieve–update pathway.

Broader benchmarks such as MLE-bench (Chan et al., 2024), MLAgentBench (Huang et al., 2024), and RE-Bench (Wijk et al., 2024) measure ML experimentation and AI R&D against quantifiable objectives. AIDE (Jiang et al., 2025), DS-Agent (Guo et al., 2024), and SELA (Chi et al., 2024) automate data-science and modeling pipelines; ADAS (Hu et al., 2024) and Darwin Gödel Machine (Zhang et al., 2025) improve agent systems or code. Hyperagents (Zhang et al., 2026a) makes the meta-level modification procedure editable, Frontis-MA1 (Yang et al., 2026) trains a dedicated meta-evolution agent, and the Red Queen Gödel Machine (Iacob et al., 2026) co-evolves agents and evaluators so the objective changes across epochs.

These systems make automated engineering measurable, but generally start from an already operationalized task, score, or evaluator. Complementary shadow evaluations find that frontier agents can complete engineering without advancing an unpublished research question (Kirgis et al., 2026), and a survey of 1,250 self-improvement papers identifies research direction-setting as the bottleneck that keeps humans in the loop (Chen et al., 2026a). This matters when an existing metric saturates or no longer represents the next capability gap. ASPIRE withholds the downstream task and measures whether an agent can turn a broad direction into learning measured on hidden evaluation items: RQ1 isolates this goal-side difference within PostTrainBench, and RQ2 studies weight evolution directed by external models or the base model itself.

Memory, workflow, and skill evolution. Self-improvement can also occur through memory, tools, and workflows. Evo-Memory (Wei et al., 2026) studies writing, organizing, and reusing memory; EvoFSM (Zhang et al., 2026b) uses interpretable finite-state workflow edits; and SkillLearnBench (Zhong et al., 2026) finds that external feedback supports continual skill gains whereas self-feedback alone can cause recursive drift. HELIX (Fan & Huang, 2026) rebuilds a fixed model's harness, updates the model from verified trajectories, and rebuilds the harness again; Co-Harness (Chen et al., 2026b) alternates validated harness edits with fine-tuning on trajectories produced by the improved harness. RQ3 connects this line to vague-goal learning by asking whether the harness mediating goal interpretation, tool use, memory, and self-evaluation can evolve while runtime weights remain fixed.

5 CONCLUSION

Modern LLM agents can search for effective ways to improve a specified objective. The next step is to determine what should be learned when no agent-visible task specification or decomposable reward has already operationalized progress. We formalize this problem as vague-goal-driven self-evolution and introduce ASPIRE, which pairs vague goals with hidden evaluation and auditable trajectories. Across three studies, the RQ1 process slice associates vague-goal prompting with more goal-definition work; self-directed runs produce evaluated checkpoints far more often than retained improvements; and the highest successor-harness mean remains below a fixed reference under unchanged runtime weights. These results show why progress must be judged against the base model rather than against the previous checkpoint: closing the training loop is not yet the same as closing the capability loop.

Our conclusions are bounded by the six goals and the coverage and scoring quality of their expert-authored evaluation items, one canonical adaptive-feedback run per configuration–goal cell, and content-level trace evidence available only under controlled access. Broader goal coverage, repeated runs, refreshed evaluation-set versions, and recursive evolution that controls drift and unrelated capability loss remain important directions for future work.

AI USE STATEMENT

Generative AI is part of the experimental systems studied in this paper: decision models select data and direct updates, models may synthesize training or validation content, several models screen candidate evaluation items for difficulty, and model-based judges score open-ended responses under fixed rubrics. These roles and information boundaries appear in Section 2 and Appendices A and C.

Beyond these experimental systems, we used generative AI tools to implement methods (including code for the environment, training, and evaluation pipelines), to design and provide feedback on the research methodology and experiments, to propose and refine hypotheses, to support qualitative analysis of agent trajectories and the interpretation of results, and to assist with translation. We did not use generative AI tools to develop the conceptual framework, to generate synthetic datasets outside the experimental systems described above, or to clean and reformat datasets; formulating mathematical claims and writing proofs are not applicable, as the paper contains no theorems. Additionally, we used generative AI tools to draft and edit parts of the paper, to create and modify figures, to search for and summarize related literature, and to convert the manuscript to the conference template, including condensing and reorganizing prose, adapting figure placement, and formatting references. We have reviewed all AI-assisted work: the authors checked all AI-assisted text, numerical values, and citations against the experimental records and cited sources, and reviewed and tested AI-assisted code. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 2 defines the task, information boundary, and retention rule. Appendices A, B, and C document evaluation construction, prompt contracts, pairing, run counts, budgets, scoring, and selection. Appendix D describes the scope of qualitative trace evidence. Content-bearing evaluation assets and traces have controlled access; the portable accounting artifact excludes hidden items and raw model messages, as specified in Appendix C.

ETHICS STATEMENT

The benchmark includes medical and other expert-domain tasks; its aggregate scores do not establish clinical reliability or readiness for deployment. The environment separates agent-directed learning from hidden measurement, checks training-data overlap, and verifies candidate provenance before scoring. Safe retention is score-based and does not certify safety or preservation of unrelated capabilities. Releasing evaluation content and raw traces requires care to preserve the benchmark's information boundary.

REFERENCES

Aditya Challapally, Chris Pease, Ramesh Raskar, and Pradyumna Chari. The GenAI divide: State of AI in business 2025. https://www.artificialintelligence-news.com/wp-content/uploads/2025/08/ai_report_2025.pdf, 2025. Industry report.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. MLE-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

Mingguang Chen, Licheng Wang, and Bo Qu. Recursive self-improvement in AI: From bounded self-refinement to autonomous research loops, 2026a. URL https://arxiv.org/abs/2607.07663.

Zhengyu Chen, Teng Xiao, Huaisheng Zhu, Yige Yuan, Luan Zhang, and Jingang Wang. Co-Harness: Co-evolving harnesses and model weights for LLM agents, 2026b. URL https://arxiv.org/abs/2607.22688.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. In *International Conference on Machine Learning (ICML)*, 2024.

Yizhe Chi, Wenyi Li, Deyao Hong, Xiaoqiu Wang, Mingju Gao, Kaisen Yang, Bingxiang He, Youjie Zheng, Calvin Xiao, and Qinhuai Na. AI4AI-Bench: Benchmarking LLM agents in algorithmic design for recursive self-improvement, 2026. URL https://arxiv.org/abs/2608.20318.

Yizhou Chi, Yizhang Lin, Sirui Hong, Duyi Pan, Yaying Fei, Guanghao Mei, Bangbang Liu, Tianqi Pang, Jacky Kwok, Ceyao Zhang, Bang Liu, and Chenglin Wu. SELA: Tree-search enhanced LLM agents for automated machine learning. *arXiv preprint arXiv:2410.17238*, 2024.

Tianyu Fan and Chao Huang. HELIX: Model-harness co-evolution for recursive self-improvement, 2026. URL https://arxiv.org/abs/2608.13951.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. DS-Agent: Automated data science by empowering large language models with case-based reasoning. In *International Conference on Machine Learning (ICML)*, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-Zero: Self-evolving reasoning LLM from zero data. *arXiv preprint arXiv:2508.05004*, 2025.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. In *International Conference on Machine Learning (ICML)*, 2024.

Alex Iacob, Andrej Jovanović, William F. Shen, Daniel Burkhardt, Meghdad Kurmanji, Nurbek Tastan, Lorenzo Sani, Niccolò Alberto Elia Venanzi, Ambroise Odonnat, Zeyu Cao, Bill Marino, Xinchi Qiu, and Nicholas D. Lane. The Red Queen Gödel Machine: Co-evolving agents and their evaluators, 2026. URL https://arxiv.org/abs/2606.26294.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. AIDE: AI-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *arXiv preprint arXiv:2009.13081*, 2020.

Peter Kirgis, Sayash Kapoor, Andrew Schwartz, Stephan Rabanser, David Africa, Konstantinos Voudouris, Viet Nguyen, Toby Pilditch, Magda Dubois, Harry Coppock, Cozmin Ududec, Nitya Nadgir, Matilda Orona, Tilman Bayer, Derrick Chan-Sew, Yue Ling, Abhishek Shetty, Helen Toner, Gillian Hadfield, Seth Lazar, Steve Newman, Shoshannah Tekofsky, Rishi Bommasani, and Arvind Narayanan. Can AI agents conduct open-ended AI research? Early evidence from two case studies, 2026. URL https://arxiv.org/abs/2607.27191.

Fanqing Meng, Lingxiao Du, Qiguang Chen, Ziqi Zhao, Haocheng Lu, Mengkang Hu, and Michael Qizhe Shieh. RSIBench-Data: Benchmarking data-centric research for recursive self-improvement, 2026. URL https://arxiv.org/abs/2607.25886.

Gergely Orosz. What are forward deployed engineers, and why are they so in demand? https://newsletter.pragmaticengineer.com/p/forward-deployed-engineers, 2025. The Pragmatic Engineer.

Palantir Technologies. A day in the life of a Palantir forward deployed software engineer. https://blog.palantir.com/a-day-in-the-life-of-a-palantir-forward-deployed-software-engineer-45ef2de257b1, 2020.

Qwen Team. Qwen-Agent: Agent framework and applications built upon Qwen. GitHub repository, 2023. URL https://github.com/QwenLM/Qwen-Agent.

Qwen Team. Qwen3.5: Towards native multimodal agents. Blog post, 2026. URL https://qwen.ai/blog?id=qwen3.5.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. PostTrainBench: Can LLM agents automate LLM post-training? *arXiv preprint arXiv:2603.08640*, 2026.

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. *arXiv preprint arXiv:2311.12022*, 2023.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research (TMLR); arXiv:2312.06585*, 2024.

The New Stack. Why OpenAI and Anthropic are hiring forward deployed engineer teams. https://thenewstack.io/forward-deployed-engineers-ai/, 2026.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-Memory: Benchmarking LLM agent test-time learning with self-evolving memory, 2026. URL https://arxiv.org/abs/2511.20857.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Holden Karnofsky, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*, 2024.

Shuhan Xue, Zixin Ding, Yichen Shen, Yinjie Wang, Zhenfei Yin, Yingcheng Wu, Yuxin Chen, Mengdi Wang, and Ling Yang. PAST-Bench: Benchmarking the foundations of recursive self-improvement in personal agents, 2026. URL https://arxiv.org/abs/2608.04003.

Junlin Yang, Che Jiang, Yu Fu, Tianwei Luo, Can Ren, Weizhi Wang, Kaikai Zhao, Hongyi Liu, Yuxin Zuo, Yuru Wang, Yuchen Fan, Kai Tian, Zhenzhao Yuan, Xiaojian Lin, Li Sheng, Rushi Qiang, Guoli Jia, Xingtai Lv, Ermo Hua, Dianqiao Lei, Youbang Sun, Ning Ding, Bowen Zhou, and Kaiyan Zhang. Frontis-MA1: Training an AI4AI model towards recursive self-improvement in machine learning engineering, 2026. URL https://arxiv.org/abs/2607.28568.

Taro Yano, Yoichi Ishibashi, and Masafumi Oyamada. LaMDAgent: An autonomous framework for post-training pipeline optimization via LLM agents, 2025. URL https://arxiv.org/abs/2505.21963.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. In *International Conference on Machine Learning (ICML)*, 2024.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel Machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025.

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

Shuo Zhang, Chaofa Yuan, Ryan Guo, Xiaomin Yu, Rui Xu, Zhangquan Chen, Zinuo Li, Zhi Yang, Shuhao Guan, Zhenheng Tang, Sen Hu, Liwen Zhang, Ronghao Chen, and Huacan Wang. EvoFSM: Controllable self-evolution for deep research with finite state machines, 2026b. URL https://arxiv.org/abs/2601.09465.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute Zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025.

Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo F. R. Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. SkillLearnBench: Benchmarking continual learning methods for agent skill generation on real-world tasks, 2026. URL https://arxiv.org/abs/2604.20087.

Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. *arXiv preprint arXiv:2506.10943*, 2025.

A Benchmark Construction and Experimental Contracts

A.1 Goal and Evaluation-Set Statistics

The benchmark defines six vague goals, each paired with one mutually exclusive group of items in the hidden evaluation set (Table 2). The adaptive-feedback protocol evaluates all six goals. Logic, reliability, and instruction following form one composite goal in the current release; we do not treat its three components as separately identified goals.

Table 2: The six vague goals and their corresponding evaluation items. All 520 top-level items are newly authored by domain experts.

Vague goal / evaluation slice	Items
Scientific and academic reasoning	75
Humanities and social-science knowledge	110
Health and medical reasoning	100
Mathematical reasoning	126
Logic, reliability, and instruction following	89
Academic and scientific writing	20
Total unique items	520

The items for the academic and scientific writing goal comprise 20 top-level task bundles, counted as 20 evaluation items. Its harness evaluator may produce multiple scored examples within a task, which is why RQ3 reports both task-macro and example-micro aggregation.

A.2 Contamination-Control Details

Contamination control begins at authorship. Domain experts write each item from scratch, while named public benchmarks serve only as task-archetype and difficulty references. Before freezing a version of the hidden evaluation set, the pipeline removes exact and semantic duplicates within and across reporting groups and audits overlap against the reference benchmarks. Dataset registration then applies a second exact and semantic overlap gate against the hidden evaluation set, and detected matches are rejected before data can enter SFT, GRPO, or continual-pretraining during a campaign.

Seed-2.0, GPT-5.2, and Gemini-3 independently answer each candidate under a shared blind-screen protocol. L0–L3 denote the number of models that fail; timeouts and service errors are retried rather than counted as failures. This screen calibrates difficulty and is separate from the authorship and overlap audits. After review, scorer binding, and local end-to-end validation, the hidden evaluation set is frozen under a manifest and SHA-256. Corrections create a new version; a running campaign continues to use the version fixed at creation.

A.3 Prompt Contracts

Prompts are rendered from a versioned setting profile rather than written ad hoc for each run. A vague-goal prompt contains a natural-language capability goal, the model being trained, the action contract, budgets, and the rule that the hidden evaluation set returns aggregate scores only. It does not reveal evaluation item content, answers, rubrics, goal-routing identifiers, or per-item feedback. For example, the mathematical goal asks the agent to improve difficult multi-step reasoning, precise calculation, and clear justification without naming the evaluation items.

An explicit-task contract additionally discloses the approved task format, difficulty anchor, metric, and success criterion, while preserving the same content boundary. Every campaign binds its setting identifier, prompt-profile version, rendered prompt SHA-256, and experiment identity. These bindings are checked on creation and recovery so that a vague-goal run cannot silently resume under an explicit-task contract or a different prompt version.

B Complete RQ1 Score and Trajectory Accounting

Score comparison. Table 3 gives the exact values visualized in Figure 2(a). Official scores are system-level PostTrainBench references; the vague-goal columns are our results. They share the benchmark definition and aggregation weights, but are not treated as prompt-only causal pairs.

Table 3: PostTrainBench score comparison for RQ1 (percent). Official columns are system-level references. Weighted means use official benchmark weights.

	Claude Opus 4.8		GPT-5.6	
Benchmark	Official Max	Vague	Official	Vague
AIME 2025	10.83	5.83	7.50	3.75
GPQA Main	28.35	24.89	30.13	27.04
HealthBench	31.68	12.62	27.39	9.93
HumanEval	47.33	56.63	63.95	56.40
GSM8K	68.87	66.98	69.48	74.35
ArenaHard	36.09	13.07	26.40	21.72
BFCL	47.13	58.63	94.38	79.38
Weighted average	32.90	27.07	36.23	29.58

Matched process accounting. The process comparison pairs explicit-task and vague-goal runs with the same decision model, base model, benchmark leaf, and trial index. We exclude anti-cheat hits and run pairs that return only the fallback base. The matched Claude Opus 4.8 comparison contains 48 matched run pairs. Each trial occupies one H20 sandbox GPU for at most 10 hours. We partition occupancy into decision-model waiting time, during which the sandbox GPU is idle, and active time used by the remaining workflow.

For Claude Opus 4.8, vague-goal prompting is associated with 2,109 more seconds of decision-model thinking and 0.61 more hours of idle time per matched run pair, alongside 1.27 fewer active hours. The corresponding absolute means move from 0.6 to 1.2 idle hours and from 8.0 to 6.7 active hours. Figure 2(b) converts these deltas to minutes for a common visual scale; displayed values use the unrounded measurements.

Behavioral normalization. Narrative traces differ substantially in length across conditions, so action mentions are normalized per 10,000 characters and interpreted only when hit rate and density move consistently. Under vague-goal prompting, the density of reading task definitions rises by $2.98\times$, inspecting evaluation scripts by $2.39\times$, and low-rank adaptation (LoRA)/parameter-efficient fine-tuning (PEFT) use by $3.09\times$. LoRA appears in 89.8% of vague-goal runs versus 24.1% of matched explicit-task runs; 33 matched run pairs switch from not using LoRA to using it, with no switch in the opposite direction.

We separately validate the Claude Opus 4.8 pattern with trace-level tool calls on the six matched settings shared by both trace exports (21 trajectories). Mean action count changes from 188.5 to 179.2 per trajectory. We therefore interpret the longer vague-goal narrative as additional deliberation, not as evidence that the agent executed more work. Training starts are counted only from concrete launch commands (e.g., `accelerate launch` or `torchrun`); broad keyword matches over `train` and `sft` are not used as action evidence. These trace-level calls are broader than the accepted controller actions defined in Section 2.1.

Official-trajectory comparison. We apply the same trajectory parser to 112 released official traces, 56 each for the official Claude Opus 4.8 Max and GPT-5.6 (Sol) systems. Vague-goal Claude Opus 4.8 averages 3.54 evaluations and 6.17 training starts per trajectory; official Claude Opus 4.8 Max averages 2.89 and 5.14, while official GPT-5.6 averages 39.21 and 14.91. These correspond to 0.57, 0.56, and 2.63 evaluations per training start. Official Claude Opus 4.8 Max additionally records 165,851 thinking characters per trajectory, whereas official GPT-5.6 records 30,895, supporting the deep-deliberation and high-frequency-experimentation interpretations in Appendix F.1.

C Complete RQ2 Results and Experimental Protocol

C.1 Outcome Accounting and Rollback

Only checkpoints whose parent, registered data, admitted update, and published artifacts are verified enter the evaluated lineage. The campaign fixes its hidden evaluation set, scorer or judge, decoding and aggregation rules, query allowance, eligibility requirements, terminal selector, and tie-breaking order before execution.

An *evaluated checkpoint* is any verified candidate with a durable score. An *eligible checkpoint* additionally satisfies the predeclared provenance, exploration, completion, and terminal-record requirements. The *selected checkpoint* is the highest-scoring eligible checkpoint under the fixed selector and tie-breaking order. The *retained model* uses that checkpoint only when it exceeds the base score; otherwise the controller rolls back to the base. We report the best evaluated, selected, and retained outcomes separately because a high intermediate score may come from an incomplete run, while a nonnegative retained score may reflect rollback rather than learning. RQ3 instead freezes each harness before research evaluation, compares it with a fixed-weight reference, and withholds the score from the creator.

C.2 Feedback and Submission Protocol

Vague-goal and explicit-task configurations use separate versioned settings and cannot submit checkpoints across experiments. Training uses registered data and a controller-resolved parent, and only a completely published checkpoint from the current lineage can be validated or submitted.

The agent's own validation data are agent-visible and do not consume queries to the hidden evaluation set. The adaptive-feedback protocol permits repeated aggregate-score queries on the same goal-specific evaluation items within its recorded allowance. The final-only protocol returns no intermediate score and accepts one final-checkpoint evaluation per run. Invalid or duplicate requests do not consume a query; an accepted reservation remains counted if a later deployment or evaluator stage fails. No protocol returns hidden items, answers, candidate outputs, rubrics, or judge reasoning.

C.3 Final-Only Protocol: Vague-Goal Weight Evolution

The final-only protocol crosses Qwen3.5-4B Self and Qwen3.5-9B Self with six goals, each with separately executed Run A and Run B, for 24 runs in total (Table 4). Each run may launch multiple training jobs within 40 GPU-hours but may submit only one final checkpoint for evaluation on the hidden evaluation set. Mean is the arithmetic mean of the two final-checkpoint scores, not best-of-two or two samples from one checkpoint. The evaluator runs as a local process, checkpoint inference uses no-thinking mode, and `gemini-3.5-flash` judges under an 8,192-token thinking budget and a 10% fail-closed threshold. Deployment uses eight GPUs and is released after evaluation.

All 24 submitted final checkpoints complete evaluation on the first judge attempt; all 2,080 item outputs are nonempty and no item reports an evaluation error. At the model–goal reporting level, Qwen3.5-4B has 0/6 Mean outcomes above its base score and Qwen3.5-9B has 1/6. At the individual-run level, the corresponding counts are 0/12 and 3/12. Scientific and academic reasoning is the only positive Mean: its two runs each score 48.00 and answer 36/75 items correctly, yet only 26 correct items overlap and 20/75 items flip which run is correct. The agreement therefore concerns aggregate direction on the same 75 evaluation items, not item-level stability or transfer beyond those items. The raw final-checkpoint macro average falls from 35.735 to 18.609 for 4B and from 39.010 to 25.640 for 9B; rollback leaves the retained model equal to the base model in 21/24 runs.

Five submitted checkpoints use numeric-label MMLU SFT. Their selected training snapshots contain 21,000 single-digit targets, and all 279 corresponding evaluation outputs are also single digits; the five final-checkpoint scores are 0, 0, 0, 6.141, and 0. Three long-tail runs further illustrate long blind training lineages: 4B Run B for mathematics launches 20 jobs and spends 26.570 GPU-hours before submitting a final checkpoint that scores 0.159; 9B Run B for mathematics launches 17 jobs and spends 32.286 GPU-hours before scoring 2.540; and 9B Run B for writing launches 16 jobs and spends 30.530 GPU-hours before scoring 0. Because only final checkpoints are evaluated, these results neither identify the best intermediate checkpoint nor establish a monotonic degradation path.

Table 4: Final-only terminal scores before rollback, two runs per goal. Mean and Δ use unrounded scores; goal names follow Table 2.

Qwen3.5-4B Self					
Goal	Base	Run A	Run B	Mean	Δ
Science	44.000	0.000	0.000	0.000	-44.000
Humanities	18.820	9.000	5.909	7.455	-11.365
Mathematics	17.860	16.746	0.159	8.452	-9.408
Medicine	87.000	63.000	68.000	65.500	-21.500
Logic / reliability	27.460	21.437	23.754	22.596	-4.864
Writing	19.270	0.000	15.308	7.654	-11.616
Qwen3.5-9B Self					
Goal	Base	Run A	Run B	Mean	Δ
Science	45.330	48.000	48.000	48.000	**+2.670**
Humanities	27.450	10.273	9.091	9.682	-17.768
Mathematics	25.160	0.714	2.540	1.627	-23.533
Medicine	87.000	71.000	72.000	71.500	-15.500
Logic / reliability	26.190	6.141	16.338	11.240	-14.950
Writing	22.930	23.588	0.000	11.794	-11.136

C.4 Adaptive-Feedback Protocol Results

Of the 30 configuration–goal cells in the adaptive-feedback protocol, 28/30 produce at least one evaluated checkpoint. Only two configuration–goal cells have a best evaluated checkpoint whose score exceeds the base score: Qwen3.5-4B Self science rises from 44.00 to 45.33, while Terra optimizing Qwen3.5-4B mathematics rises from 17.86 to 20.10. The Terra mathematics checkpoint is the only retained improvement under the complete-run selection rule. Among the 22 configuration–goal cells with at least two evaluated checkpoints, 62 consecutive transitions comprise 28 increases, 13 ties, and 21 decreases. Later search beats the first evaluated checkpoint in 14/22 configuration–goal cells, but only the Terra mathematics trajectory yields an eligible checkpoint whose score exceeds the base score. These are descriptive cell-level results from one run per configuration–goal cell.

Decision-model search profiles. Eight of the 22 multi-checkpoint trajectories are non-decreasing. Luna uses SFT in 24/28 registered plans and continues from the latest checkpoint in 21/28, producing 22 evaluated checkpoints with 89.57 settled GPU-hours. Terra registers 33 plans, including six GRPO updates, and produces 19 evaluated checkpoints with 91.73 GPU-hours. Its retained mathematics score of 20.10 is followed by 2.78 and then 18.17. Sol registers 40 plans, 36 evaluation attempts, 23 search actions, and 12 distinct dataset names; it produces 33 evaluated checkpoints with 76.56 GPU-hours, but none exceeds the corresponding base score.

Self-directed proxy and update choices. Qwen3.5-4B Self imports GSM8K or Hendrycks mathematics data in 30/32 dataset-import events, including events for science, logic, and writing; Qwen3.5-9B Self uses mathematics data in 11/15 imports but is more goal-specific outside mathematics. Their update choices differ as well: Qwen3.5-4B Self uses GRPO in 4/12 plans, whereas Qwen3.5-9B Self uses SFT in 9/11. The two configurations generate 20,396 and 28,135 agent events but register only 12 and 11 training plans, respectively. These counts describe search allocation and do not by themselves establish why a checkpoint gains or loses capability.

C.5 Trajectory Evidence Boundary

The portable adaptive-feedback artifact contains counts, timestamps, source hashes, resource use, submissions, and selection records, but excludes hidden evaluation items, raw model messages, command outputs, and per-item judge reasons. Deterministic extraction covers every configuration–goal cell, reading 63,973 agent, 1,196 training, and 2,194 phase events and recovering 124 training plans and 95 candidate-evaluation attempts. Two independent reviews check the task-level interpretation.

The resulting claims therefore concern exploration, checkpoint progression, selection, and rollback; item-level causal analysis requires controlled access to the content-bearing traces.

C.6 Compute and Resource Accounting

Table 5: Resource accounting for RQ2–RQ3. Training GPU-hours exclude inference, judging, deployment, and idle controller time.

Experiment	Runs	Plans	Jobs	Train GPU-h
Adaptive feedback	30	124	107	322.610
Final-only	24	—	83	149.236
Harness creation	4	—	—	0

Adaptive feedback	One run/configuration–goal cell; 40 GPU-h and 10 h wall time/cell.
Final-only	Separately executed Run A and Run B/model–goal pair; 40 GPU-h/run.
Harness creation	3 valid successors; three executions/successor harness.

Table 5 summarizes runs, plans, jobs, and settled training GPU-hours. The adaptive-feedback protocol is a complete 5×6 design but not a multi-seed estimate. The final-only protocol uses 59.269 training GPU-hours for Qwen3.5-4B and 89.967 for Qwen3.5-9B. RQ3 does not update weights; creator-model and runtime inference therefore lie outside the training-GPU ledger.

D RQ3 Creator-Trajectory Case Analysis

Scope and evidence boundary. This content-level analysis covers the three valid GPT-5.6 creator trajectories and one inspected execution on the hidden evaluation set for each frozen successor harness. The creator sees only its own validation data; evaluation items and scores are not returned. Table 1 reports means over three executions, whereas the task-level examples below diagnose mechanisms in one inspected execution. They are therefore case evidence, not additional repetitions or a stable ranking of creator models.

Luna: early stopping on a narrow validation proxy. Luna evaluates H_0 on one validation prompt with an eight-item checklist. After a 7/8 result, it adds a fixed five-part template covering the research question, design, analysis, risks, and limitations. The edited harness reaches 8/8 on that prompt and two similar prompts, and Luna stops after approximately 40 minutes. In the inspected execution on the hidden evaluation set, this template improves structural completeness but reframes direct questions about mechanisms, formulas, parameters, or engineering details as study-design problems rather than supplying the requested technical content.

Terra: reviewer reversion without a final-answer invariant. Terra removes an initial two-pass reviewer after it invents unsupported significance results and effect sizes, then adds narrower safeguards for assumptions, model calls, logging, and calculator use. It nevertheless omits a final-answer invariant: when the post-calculator response is empty, the harness submits the pre-tool fragment. Two inspected tasks therefore return partial drafts of 342 and 520 characters, with scores falling from 75.00 and 52.33 under H_0 to 2.57 and 2.91. These cases account for approximately 70% of the aggregate decline in that execution.

Sol: broader testing followed by conservative selection. Across 3 hours 11 minutes and four validation task types, Sol tests prompt length, tool-call recovery, disabled tools, and a review pass, while correcting one erroneous local reference answer. It rejects locally stronger variants for latency, repeated calls, fabricated numbers, or scorer misses. The selected harness restores the original Qwen-Agent policy and adds a completeness check for empty final responses. The check never triggers on the 20 inspected evaluation items, supporting improved failure handling, not a score gain.

E ADDITIONAL TASK AND ENVIRONMENT DETAILS

Self-evolution can only be studied if progress remains externally measurable. ASPIRE therefore keeps an evaluator as controller-side scientific instrumentation while withholding its benchmark definition, items, and decomposable reward from the agent. Bounded aggregate outcomes approximate sparse deployment feedback without turning the evaluator into an agent-visible task specification or source of directly trainable supervision.

This section instantiates that setting through three components. We first define a bounded evolution episode and the model and harness surfaces on which it may operate (Section 2.1). We then present a hidden evaluation set spanning six goals (Section 2.2) and the minimal interactive environment through which an agent constructs, tests, and safely retains updates (Section 2.3). Experiment-specific feedback, eligibility, resource, and release accounting appears with the RQ2 results in Appendix C.

E.1 ROUNDS AND EVOLUTION SURFACES

An evolution round is a bounded search-and-commit episode during which the decision model is fixed, rather than one tool call. Formally, $D_{r,j} = D_r$ for every interaction step j in round r. Together with a fixed creator-side scaffold C_r, D_r interprets G and may propose multiple data operations, updates, validations, branches, and candidate states before termination. Different candidates therefore embody different goal operationalizations. After the round terminates, the controller may reuse D_r or explicitly promote a verified trained descendant. Writing $\mathcal{D}_r$ for the set of such descendants, the next-round choice obeys $D_{r+1} \in \{D_r\} \cup \mathcal{D}_r$; any handoff begins only in the subsequent round.

Released interaction. Let $E_{r,j}$ be the private controller state before interaction step j. The agent receives only a released view $q_{r,j}$ containing the public history, lifecycle status, legal request types, approved identifiers, and a bounded projection of the remaining budget. At each step, D_r and C_r propose an action from the goal and released history, and the controller validates the request before updating its private state. Accepted requests create the corresponding durable transition; rejected requests return a typed error without creating an artifact or score. Detailed feedback is restricted to the agent's own validation data; evaluation feedback follows the protocol-specific information boundary in Section 2.2.

Two evolution surfaces. ASPIRE separates the component being evolved from the policy directing the search. The two reported surfaces are

$$\mathcal{S}_M(H_0, D_r) = \{(M, H_0, D_r) : M \in \mathcal{M}\}, \qquad \mathcal{S}_H(M_0, D_r) = \{(M_0, H, D_r) : H \in \mathcal{H}\}.$$

Table 6: Evolution surfaces in ASPIRE. The component under study changes while D_r, the controller, and the evaluator remain fixed within the round.

Setting	Mutable component	Fixed during evaluation
Weight evolution (RQ1–RQ2)	Model weights M	H_0, D_r, controller, evaluator
Harness evolution (RQ3)	Agent harness H	M_0, D_r, controller, evaluator

E.2 EVALUATION CONSTRUCTION AND INTERPRETATION

This subsection describes the six-goal hidden evaluation used in RQ2 and RQ3; RQ1 instead uses the original sealed, task-specific PostTrainBench evaluators under a vague-goal contract, without pooling those results into ASPIRE.

Items for six goals. The 520 items cover the six goals: scientific and academic reasoning (75 items); humanities and social-science knowledge (110); health and medical reasoning (100); mathematical reasoning (126); logic, reliability, and instruction following (89); and academic and scientific writing (20) (Figure 6). The fifth goal is intentionally composite; the current protocol does not claim to measure logical reasoning, hallucination resistance, and instruction following as three independently identifiable targets. Instead, it treats them as one integrated reliability objective. Each evaluation item

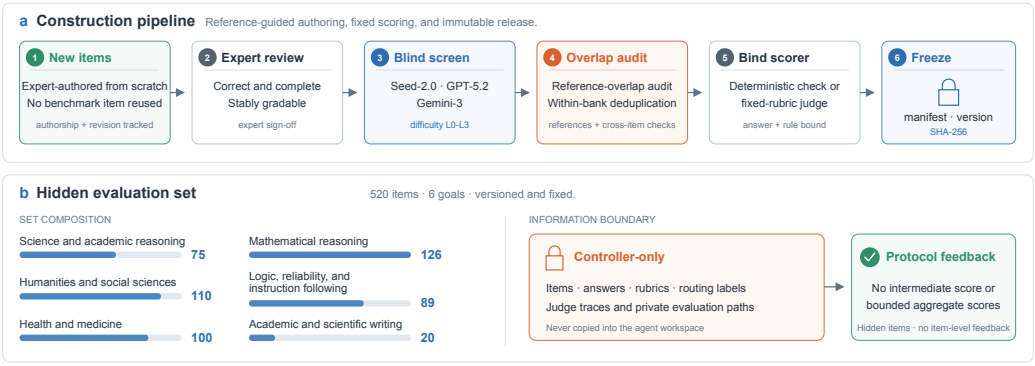

Figure 6: Hidden evaluation set. **Top:** expert-authored candidates pass independent review, blind difficulty screening, overlap and duplication audits, scorer binding, and immutable versioning. **Bottom:** 520 items over six goals; items and judging assets remain controller-only.

belongs to exactly one goal, and each goal is scored and reported on its own evaluation slice rather than pooled into a single item-level score across all six goals. The writing items are 20 top-level task bundles that may yield multiple evaluator-scored responses, so RQ3 reports task-macro and example-micro aggregations separately.

Construction and quality control. Domain experts author every candidate item from scratch. GPQA (Rein et al., 2023), MMLU-Pro (Wang et al., 2024), and MedQA (Jin et al., 2020) serve only as references for task format, domain coverage, and approximate difficulty; no item from these benchmarks is copied, rewritten, or included. Each candidate retains authorship and revision provenance, a reference answer or fixed rubric, and a predefined scoring rule. Independent review removes incorrect, incomplete, underspecified, or unstably gradable items.

Surviving candidates pass blind multi-model difficulty calibration, exact and semantic deduplication, overlap auditing against the reference benchmarks, scorer binding, and local end-to-end validation. Deterministic or structured scoring is used where possible; open-ended responses use a campaign-fixed rubric-based judge. The hidden evaluation set is bound to an immutable versioned manifest, and a running campaign never changes that version. Detailed screening, difficulty levels, overlap checks, scorer assignments, and versioning appear in Appendix A.1 and Appendix A.2.

We use *out-of-distribution (OOD)* operationally to describe evaluation items that are independently authored, excluded from the agent's training data and its own validation data, and evaluated outside the learning signals constructed by the agent. This does not assert that their marginal text distribution is disjoint from a model's pretraining corpus. The protocol is designed to reduce direct content leakage and to test whether an agent can operationalize a vague goal from sparse outcome feedback while preserving an auditable separation between learning artifacts and measurement artifacts.

E.3 Execution Interface and Retention

An agent-facing execution tool. The agent does not operate a shell, download a training repository, assemble a runtime, or implement distributed execution. Instead, ASPIRE exposes a single agent tool with typed, composable actions (Figure 7). Through tool calls, the agent can search for, download, or import public datasets; synthesize and register new data; launch supervised fine-tuning (SFT) or Group Relative Policy Optimization (GRPO) with low-rank adaptation (LoRA) or other permitted configurations; query job state; construct and run checks on its own validation data; and branch or stop a lineage. The agent still chooses the data, update method, parameters, and control flow, while the controller translates those choices into managed backend operations. This design minimizes incidental engineering work without removing the strategic decisions that self-evolution from a vague goal is intended to test.

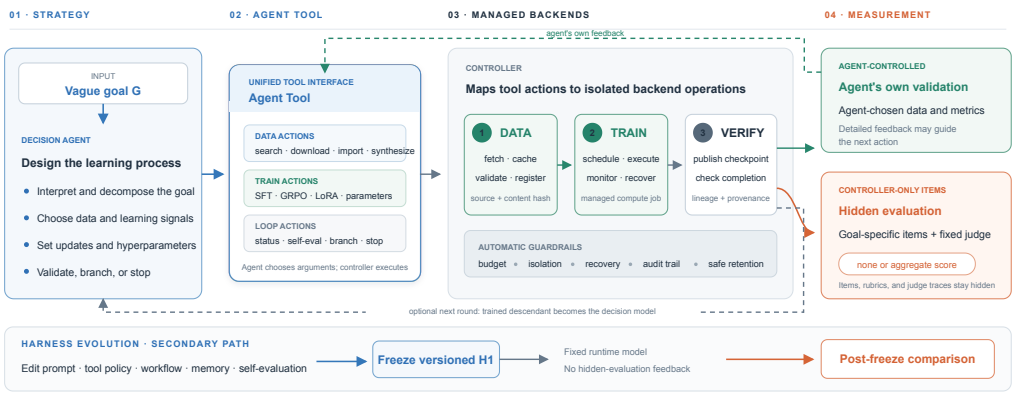

Figure 7: The interactive environment. Weight candidates follow a data–train–verify path and harness candidates an edit–freeze path. A unified agent tool exposes data, training, status, and validation actions; the controller maps them to managed backends, enforces budget and lineage constraints, and returns only the aggregate score the protocol permits.

Verified candidates and auditable execution. A weight candidate is evaluable only after the controller verifies its parent, registered data, admitted update, and complete checkpoint publication. A harness candidate must be versioned and frozen before evaluation. Invalid, incomplete, foreign, or agent-invented artifacts cannot enter the scored lineage. Accepted actions record provenance, candidate lineage, resource use, aggregate submissions, and terminal selection. Full action schemas, recovery, and backend implementation details are part of the system artifact; the experiment-facing accounting rules appear in Appendix C.

F Additional Experimental Analysis

F.1 RQ1: Reference Profiles and Summary

Question. How do vague goals change post-training outcomes and search trajectories relative to an explicit-task setting?

Reference comparison: strong explicit-task systems use different search profiles. Claude Opus 4.8 under vague-goal prompting performs 3.54 evaluations for 6.17 training starts, or 0.57 evaluations per start. Official GPT-5.6 performs 39.21 evaluations for 14.91 starts, or 2.63 per start—$4.6\times$ the feedback density (Figure 2(c)), providing more opportunities to test, reject, and branch from updates. Official Claude Opus 4.8 Max follows a different profile: its 2.89 evaluations and 5.14 training starts yield 0.56 evaluations per start, nearly identical to Claude Opus 4.8 under vague-goal prompting, but it records 165,851 thinking characters versus 30,895 for official GPT-5.6. These cross-system statistics describe two strong explicit-task profiles rather than isolate goal vagueness: dense evaluation feedback for official GPT-5.6 and larger reasoning volume for official Claude Opus 4.8 Max.

Answer to RQ1. Relative to explicit-task references, vague goals expand the search from *how to optimize* to *what to optimize and how*. In the matched Claude Opus 4.8 runs, vague goals are associated with more goal-operationalization work and less active update time, while their aggregate scores remain below the explicit-task references.

F.2 RQ2: Checkpoint Trajectories and Summary

Question. Starting from an instruction-tuned checkpoint, can an LLM given only a vague goal determine what to learn and improve its capability by directing iterative updates to that checkpoint? This starting point more closely reflects practical self-evolution, where a generally useful model must improve itself while preserving behavior acquired through instruction tuning.

Trajectory pattern 2: more search does not mean better search. Using the same Qwen3.5-4B base model, the external decision models produce different goal-level outcomes. Among their best evaluated checkpoints, Luna is highest on humanities (7.27) and logic (20.39), Terra on mathematics (20.10) and writing (16.01), and Sol on science (30.67) and medicine (11.00). In five of six goals, even the highest of these scores remains below the base score. Sol searches most broadly, producing 33 evaluated checkpoints with 76.56 settled training GPU-hours, yet none exceeds its corresponding base score. Terra finds the sole retained improvement on the adaptive selection slice; its later mathematics descendants fall from 20.10 to 2.78 before recovering to 18.17. Greater search volume therefore does not ensure an above-base score, and continued optimization can induce regression rather than produce a better descendant, making checkpoint selection and rollback necessary (Figure 8).

In the final-only protocol, three runs continue for 16–20 training jobs and consume 26.570–32.286 GPU-hours, yet their submitted final checkpoints score only 0.159 on 4B mathematics, 2.540 on 9B mathematics, and 0 on 9B writing. Because intermediate checkpoints are not evaluated under this protocol, these trajectories do not establish monotonic degradation or identify the best earlier checkpoint; they show that blind continuation along a latest lineage can end in a poor checkpoint.

Answer to RQ2. We observe isolated above-base goal scores rather than reliable improvement. Only one of 12 final-only Avg@2 outcomes exceeds its base score, and adaptive feedback often produces locally rising trajectories that remain below base. Agents therefore complete the update loop more consistently than they preserve base performance; these goal-specific evaluations do not measure unrelated capabilities.

F.3 RQ3: Creator Trajectories and Summary

Question. With model weights held fixed, can one-step harness evolution improve the agent's research performance for the academic and scientific writing goal?

Trajectory pattern 1: narrow agent validation can specialize the harness to the wrong proxy. Luna raises one creator-designed eight-item checklist on the agent's own validation data from 7/8 to

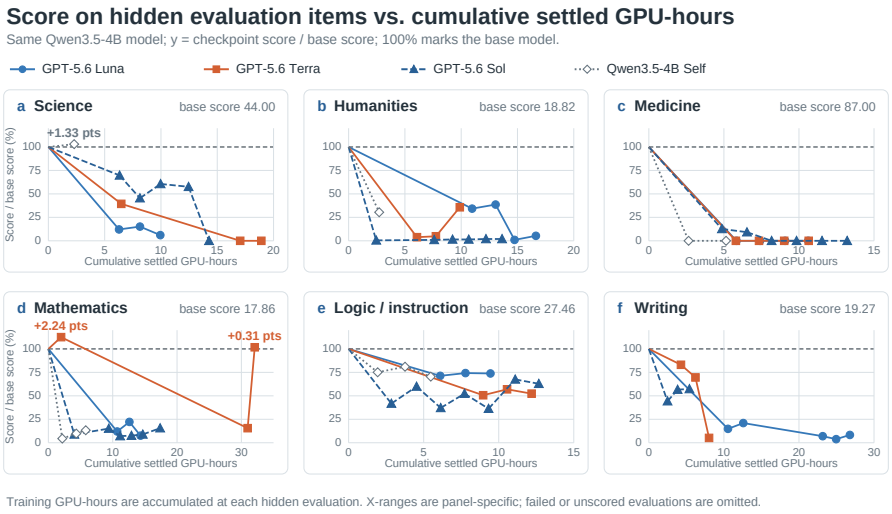

Figure 8: Adaptive-feedback score–compute trajectories for four configurations on Qwen3.5-4B. Scores are normalized to each goal's base score and plotted against settled training GPU-hours.

8/8 by adding a fixed five-part research-answer template. After two similarly structured prompts also score 8/8, it stops after approximately 40 minutes. The template makes answers more explicit about bias and limitations, but inspection of hidden evaluation items shows that it can recast requests for mechanisms, formulas, parameters, or engineering details as study-design problems. The validation gain and the 19.32 task-macro mean on the hidden evaluation set are therefore consistent with specialization to a narrow proxy rather than broad improvement.

Trajectory pattern 2: answer review does not guarantee output integrity. Terra correctly rejects a two-pass reviewer after it invents unsupported significance results and effect sizes, but its final harness retains a different failure path: after a calculator call, an empty final model response causes the pre-tool partial draft to be submitted. This occurs on two items in one inspected execution and accounts arithmetically for approximately 70% of that execution's aggregate decline. Sol searches for 3 hours 11 minutes across four task types in its own validation data and several prompt and tool variants, rejecting alternatives with hallucination, latency, or tool-control failures even when their validation scores are higher. Its final edit only checks that the last model response is complete and nonempty; the check does not trigger on any of the 20 inspected evaluation items. It therefore improves failure handling, but the trace does not show that it causes a task-accuracy gain. Detailed cases appear in Appendix D.

Answer to RQ3. One-step harness evolution can produce executable and behaviorally distinct successor harnesses, but all three valid successors remain numerically below Qwen-Agent. Their trajectories expose bottlenecks in validation diversity and output integrity: in this one-step, single-goal setting, agents complete the edit–test loop but do not beat the reference harness on the hidden set.

G BACKGROUND

Most agent benchmarks begin after the problem has already been made executable: the task is specified, the reward or judge is defined, and the execution scaffold is fixed. This setting is necessary for controlled comparison, but it hides the work that dominates real deployment. In industry, that work is distributed across several roles—including solutions architects, applied engineers, and platform engineers—but its most visible recent crystallization is the *forward-deployed engineer* (FDE), a title popularized by Palantir and since adopted by frontier-model companies (Palantir Technologies, 2020; Orosz, 2025). An FDE is embedded in the deployment environment and turns a general-purpose model into a system that works with a customer's data formats, workflows, and operational constraints. The role's success criterion is not a demonstration or a benchmark score, but whether the deployed system is genuinely used, continues to work, and improves in response to failures. The growth of FDE roles across frontier-model and data-platform companies reflects a simple fact: a capable model is not yet a working system (The New Stack, 2026; Challapally et al., 2025), and today the gap is largely closed by human engineers.

Viewed from the model's side, FDE work supplies three pieces of structure that benchmark designers normally presuppose. First, the goal is vague: an informal deployment need must be translated into concrete objectives, constraints, and success criteria. Second, the feedback signal may be absent or unreliable: tests, judges, traces, or other validation mechanisms must be constructed before anyone can determine whether the system is improving. Third, the execution system may not exist in a usable form: the tools, context management, state, lifecycle logic, and verification interface through which future tasks will run must be built, adapted, and maintained as requirements change.

ASPIRE focuses on the first layer and the capability-improvement process it initiates. When a deployment need specifies only a broad capability direction, can a model determine what it should learn, translate that direction into data, training plans, and validation signals, and achieve real capability growth by updating its model weights or agent harness? ASPIRE isolates how broad deployment needs are translated into concrete learning objectives and realized as capability growth.